IBM z/VSE
バージョン **6** リリース **2**

TCP/IP サポート

SC43-2945-01

IBM z/VSE
バージョン **6** リリース **2**

TCP/IP サポート

SC43-2945-01

お願い

本書および本書で紹介する製品をご使用になる前に、 621 ページの『特記事項』に記載されている情報をお読みください。

本書は、以下のプログラムに適用されます。

- IBM z/Virtual Storage Extended (z/VSE) バージョン 6 リリース 2 (プログラム番号 5686-VS6)
- TCP/IP for z/VSE のバージョン 2 リリース 2、プログラム番号 5686-CS1
- IPv6/VSE バージョン 1 リリース 3 (プログラム番号 5686-BS1)

上記に加え、新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本書は SC43-2945-00 の改訂版です。

資料のご注文方法については、http://www.ibm.com/jp/manuals の「ご注文について」をご覧ください。 (URL は、変更になる場合があります)

また、FAX によりまたはインターネット経由で送付することもできます。

```
Internet: s390id@de.ibm.com
FAX (Germany): 07031-16-3456
FAX (other countries): (+49)+7031-16-3456
```

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： SC34-2706-01
IBM z/VSE
Version 6 Release 2
TCP/IP Support

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

目次

図

表

本書について

この資料では、TCP/IP および接続とデータの交換を可能にするプログラムを使用して、IBM® z/VSE ホストと通信を行う方法について説明します。

この資料を使用する前に、 xix ページの『重要な考慮事項 - 最初にお読みください』セクションをお読みください。本書の内容と構成についての概要が記載されています。

関連資料

z/VSE ホーム・ページ

WWW 上に z/VSE のホーム・ページがあります。このホーム・ページでは、VSE の関連製品やサービス、z/VSE の新機能、その他 VSE ユーザーの関心が高いトピックについて最新情報を提供します。

z/VSE ホーム・ページは、次のアドレスで見ることができます。

http://www.ibm.com/systems/z/os/zvse/

次のサイトで VSE ユーザーの例 (ZIP 形式) を検索することもできます。

http://www.ibm.com/systems/z/os/zvse/downloads/samples.html

z/VSE Knowledge Center

IBM Knowledge Center は、IBM の技術情報の新しいホーム・ページです。z/VSE Knowledge Center は、次のアドレスで見ることができます。

http://www.ibm.com/support/knowledgecenter/SSB27H/zvse_welcome.html

構文図について

このセクションでは、本書の構文図の読み方を説明します。

構文図を読む場合は、線が示すパスに従って読みます。線に従って、左から右へ、また上から下に読みます。

- ▸▸── 記号は、構文図の開始を示します。
- ──▸ 記号は、行末にあり、構文図が次の行へ継続することを示します。
- ▸── 記号は、行の先頭にあり、構文図が直前の行からの継続であることを示します。
- ──▸◂ 記号は、構文図の終わりを示します。

構文内の項目 (例えば、キーワードや変数) は、次のように示されます。

- 線上に直接示される (必須)
- 線の上部に示される (デフォルト)

- 線の下部に示される (オプション)

大文字

大文字は、可能な省略形のうち最も短い省略形を示します。項目が大文字だけで示されている場合は、省略できません。

項目は、大文字だけ、小文字だけ、または大文字、小文字の任意の組み合わせで指定できます。以下に例を示します。

この例では、KEYWO、KEYWOR、または KEYWORD を大文字と小文字の任意の組み合わせで指定できます。

記号　以下の記号は、構文図に示されたとおりにコーディングしなければなりません。

*****　アスタリスク

:　コロン

,　コンマ

=　等号

-　ハイフン

//　ダブルスラッシュ

()　括弧

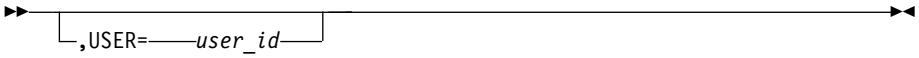
.　ピリオド

+　加算記号

以下に例を示します。

```
 * $$ LST
```

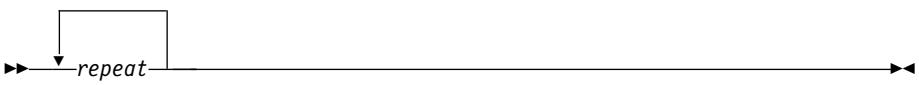
変数　強調表示された小文字は、具体的な情報で置換する必要のある可変情報を表します。以下に例を示します。



この例では、USER= はこのとおりにコーディングし、user_id には ID を指定する必要があります。USER は小文字で入力してもかまいませんが、それ以外の変更はできません。

繰り返し

左に戻る矢印は、その項目を繰り返し指定できることを意味します。



矢印内の文字は、その文字を使用して、繰り返される項目を区切る必要があることを意味します。

矢印の脚注 (1) は、その項目を何回繰り返せるかの限界を表します。

注:

1 *repeat* を 5 回まで指定。

デフォルト

デフォルト値は線の上部にあります。システムは、デフォルト値がオーバーライドされない限り、デフォルト値を使用します。線の下部にあるスタックからオプションを指定することにより、デフォルト値をオーバーライドできます。以下に例を示します。

この例では、A がデフォルト値です。B または C を選択することにより、A をオーバーライドできます。

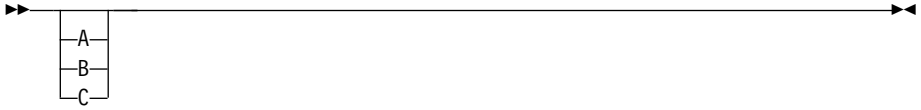
必須選択

スタックに複数の項目があり、その 1 つが線上にある場合、項目を 1 つ指定しなければなりません。以下に例を示します。

この例では、A または B または C を入力しなければなりません。

任意選択

項目が線の下部にある場合、その項目はオプションです。いずれか 1 つの項目を選択してもかまいません。以下に例を示します。



この例では、A または B または C を入力するか、このフィールドを省略できます。

必要なブランク

必要なブランクは、表記でそのように示されます。以下に例を示します。

```
 * $$ EOJ
```

この例は、文字 $$ の前後に少なくとも 1 個のブランクが必要であること
を示します。

変更の要約

この資料は、z/VSE 6.2 に実装された機能拡張および変更を反映して更新されました。また、用語、細かな修正、および編集上の変更も含まれています。

z/VSE 6.2 で利用可能になった機能拡張

- z/VSE 上の OpenSSL には、新しいランタイム・パラメーター SSL$CPH、SSL$CSI、および SSL$TRC が導入されました。
- Secure Socket Layer (SSL) というワードは、該当する場合は Transport Layer Security (TLS) に置き換えられました。
- 既知の脆弱性から、SSL30 はどのサンプルからも削除されました。SSL30 は今後使用できません。
- EZASOCKET または EZASMI のプログラミング・インターフェースの使用: OpenSSL 実装を C 以外のプログラミング言語から使用することもできます。 OpenSSL 実装を選択するには、EZA マルチプレクサーを構成する必要があります。詳しくは、 95 ページの『第 10 章 使用する TCP/IP と SSL の実装を選択』を参照してください。

z/VSE 6.1 で利用可能になった機能拡張

- z/VSE Network Appliance (VNA) ソリューション。詳細は、 485 ページの『第 16 章 z/VSE Network Appliance』を参照してください。
- Barnard Software, Inc. により提供される IPv6/VSE 製品は、z/VSE ベースの、ライブラリー PRD2.TCPIPB にプリインストールされることになりました。
- TCP/IP for z/VSE および IPv6/VSE の両方で、基本的なファイアウォール・セキュリティー機能が提供されます。詳しくは、 8 ページの『TCP/IP for z/VSE のファイアウォール』および 49 ページの『IPv6/VSE のファイアウォール』を参照してください。

z/VSE 5.2 で利用可能になった機能拡張

- Connectivity Systems Inc. から提供される TCP/IP for z/VSE 製品は、PRD1.BASE ではなくライブラリー PRD2.TCPIPC にインストールされるようになりました。そのため、現在は、TCP/IP for z/VSE を使用するジョブには、ライブラリー検索チェーンにおける最初のライブラリーとして PRD2.TCPIPC が必要となります。
- CICS リスナー・サポートは強化されて IPv6、TCP/IP スタックの仕様などもサポートしています。詳細は、 521 ページの『第 5 部 CICS リスナー・サポート』を参照してください。

重要な考慮事項 - 最初にお読みください

この資料では、TCP/IP および接続とデータの交換を可能にするプログラムを使用して、IBM z/VSE ホストと通信を行う方法について説明します。以下の TCP/IP 製品で z/VSE を実行できます。

- TCP/IP for z/VSE (5686-CS1)。IPv4 プロトコルのサポートがあります。
- IPv6/VSE (5686-BS1)。IPv4 および IPv6 プロトコルのサポートがあります。

IBM は、これらのプログラムとともに使用できる拡張機能を提供しています。ここでは、LE/VSE C ソケット・インターフェース、EZASMI マクロ・インターフェース、EZASOKET 呼び出しインターフェース、Linux Fast Path to z Systems 機能、CICS® リスナー・サポートなどの拡張機能について詳しく説明します。これらの拡張機能をその外部インターフェースで使用する際の説明には、TCP/IP スタックで実際に提供されるよりも多くの機能が含まれている場合があります。ご注意ください。そのような例外については、それぞれの機能の説明を確認してください。

この資料は、以下の 5 つの部で構成されています。

- 1 ページの『第 1 部 TCP/IP for z/VSE の使用』は、TCP/IP for z/VSE (5686-A04) 製品のセットアップおよび使用の方法を説明しています。
- 43 ページの『第 2 部 IPv6/VSE の使用』は、IPv6/VSE (5686-BS1) 製品の概要を示しています。
- 51 ページの『第 3 部 プログラミング・インターフェース』は、z/VSE ホストへのさまざまな接続方式と、システムとのデータの交換方法を説明しています。
- 443 ページの『第 4 部 Linux へのファスト・パスの使用』は、Linux on z Systems へのファスト・パスおよびz/VSE - z/VM IP アシスト機能について説明しています。
- 521 ページの『第 5 部 CICS リスナー・サポート』は、CICS リスナー・サポートの構成方法を示しています。

第 1 部 TCP/IP for z/VSE の使用

第 1 章 概要

TCP/IP は、VSE ベースのソフトウェアと、TCP/IP を備えた他のプラットフォーム上で実行するソフトウェアとの間の双方向通信を可能にする通信機能です。

TCP/IP for z/VSE プログラムをご使用になる前に、以下の項目を熟読してください。

TCP/IP for z/VSE (5686-CS1) プログラムの資料

TCP/IP for z/VSE 製品 (IBM 製品番号 5686-CS1) の製品説明は、z/VSE ホーム・ページから PDF 形式で入手できます。または IBM 資料センターからオンライン・コレクション・キット (SK3T-8348) を圧縮形式でダウンロードできます。

TCP/IP for z/VSE の製品資料には、TCP/IP for z/VSE のプロバイダーである Connectivity Systems Inc. が提供する、オリジナルの製品説明を記載した 6 種類の資料に加えて、IBM が提供する TCP/IP for z/VSE 製品のセットアップについて説明した資料があります (本書)。

それら 7 種の資料は、以下のとおりです。

- *z/VSE TCP/IP* サポート (この資料)
- *TCP/IP for VSE 2.2 Installation Guide*
- *TCP/IP for VSE 2.2 User Guide*
- *TCP/IP for VSE 2.2 Command Reference*
- *TCP/IP for VSE 2.2 Programmer's Guide*
- *TCP/IP for VSE 2.2 Messages*
- *TCP/IP for VSE 2.2 Optional Features*

Adobe Reader を使用して、これらの資料を表示および印刷できます。Adobe Reader がまだインストールされていない場合、または Adobe Reader のインストールおよび使用方法についての情報が必要な場合は、Adobe Web サイト (http://www.adobe.com) を参照してください。

z/VSE で使用する Secure Sockets Layer / Transport Layer Security (SSL/TLS) のセットアップについては、「*IBM z/VSE* 管理」で説明されています。

HiperSockets および OSA Express® のサポートの詳細については、「*IBM z/VSE* 計画」に記載されています。

TCP/IP for z/VSE プログラムのセットアップに関する一般的考慮事項

前述のとおり、TCP/IP for z/VSE 製品 (IBM 製品番号 5686-CS1) の製品資料は、z/VSE DVD コレクション (SK3T-8348) から、PDF 形式のみで入手できます。この DVD には、TCP/IP for z/VSE 製品のプロバイダーである Connectivity Systems Inc. が提供する、オリジナルの 6 種類の製品資料が収録されています。

Connectivity Systems Inc. (CSI) による製品説明をお読みになる際は、IBM 提供の TCP/IP for z/VSE 製品を使用する場合との以下の相違点にご注意ください。

- IBM 提供の TCP/IP for z/VSE 製品 (製品番号 5686-CS1) は、おおむね CSI 提供の TCP/IP for z/VSE と同じです。TCP/IP for z/VSE を活用するための追加機能および相違点は、本書で説明されています。
- Connectivity Systems Inc. から提供される TCP/IP for z/VSE 製品は、PRD1.BASE ではなくライブラリー PRD2.TCPIPC にインストールされるようになりました。このため、TCP/IP for z/VSE を使用するバッチ・ジョブは、そのライブラリー検索チェーンの最初のライブラリーとして PRD2.TCPIPC を指定する必要があります。
- IBM 提供の TCP/IP for z/VSE は、PRD2.TCPIPC ライブラリーにプリインストールされています。従って、CSI 提供の資料の中で製品のインストール作業について説明されている箇所 (製品のリストアなど) は当てはまりません。
- IBM 提供の TCP/IP for z/VSE では、固有のキー検査手順を使用します。TCP/IP for z/VSE 用の IBM 製品キーをインストールする方法については、後述します。
- TCP/IP for z/VSE で使用可能な REXX サポートには、次の 2 つのタイプがあります。
  - REXX/VSE 内部での REXX ソケット API サポート。この REXX ソケット API に関する説明は、オンライン・マニュアル「*REXX/VSE 解説書*」にあります。
  - TCP/IP for z/VSE 内部での REXX サポート (例えば、REXX ソケット API)。この REXX サポートの資料については、「*TCP/IP for VSE Programmer's Guide*」を参照してください。
- TCP/IP for z/VSE の CAF (CICS Access Facility) は、IBM からはまだ提供されていません。したがって、CAF に言及しているすべての記述は適用対象がありません。
- 最新の TCP/IP for z/VSE の APAR および PTF については、z/VSE® ホーム・ページ (http://www.ibm.com/systems/z/os/zvse/support/tcpip.html) にアクセスしてください。
- TCP/IP for z/VSE で問題が発生する場合は、Connectivity Systems Inc. の Web サイト (http://www.csi-international.com/) にある TCP/IP サポート情報を参照し、問題の解決に役立つ可能性がある入手可能なフィックスがないかどうかを確認してください。
- IBM から TCP/IP for z/VSE 製品のライセンス交付を受けている場合、問題発生時には、通常の IBM サービス・チャネルを使用してサポートを受ける必要があります。その際、問題に関するテープおよび文書を適当なサービス・センターに提供する必要があります。従って、CSI の文書に記述されている特別な技術サポート考慮事項は当てはまりません。

製品キーの提供

TCP/IP for z/VSE は、VSE 用のネイティブな TCP/IP ソリューションであり、z/VSE Base にプリインストールされています。これは、オプションで有料の IBM プログラムとして入手できます。このプログラムは、Connectivity Systems Inc. から IBM へライセンス供与されています。

基本の TCP/IP for z/VSE 機能セットである Application Pak には、キーが必要です。汎用プリント・サーバー (GPS) フィーチャーは、TCP/IP for z/VSE Application Pak に追加する、オプションで有料のアプリケーションであり、別のキーを必要とします。

SSL for VSE は、TCP/IP for z/VSE の一部であり、これもキーで保護されています。

Application Pak または GPS 用のそれぞれ異なるキーは、製品のライセンス交付を受ける際にお客様に送達されます。TCP/IP for z/VSE 製品およびそのフィーチャーのライセンスを取得するには、IBM の通常のご注文プロセス (CFSW を使用する注文プロセスなど) に従う必要があります。

Application Pak は、ソケット・アプリケーション・プログラミング・インターフェース (API)、TCP/IP プロトコル・スタックを包含し、物理層からアプリケーション関数までの TCP/IP 通信のすべての層を扱います。これには、以下の TCP/IP アプリケーションも含まれます。

- TN3270 サーバーおよび Telnet/TN3270 クライアント
- FTP サーバーおよびクライアント
- Web サーバー (HTTP デーモン)
- ライン・プリンター・リクエスター (LPR) およびライン・プリンター・デーモン (LPD)

GPS は、Application Pak に組み込まれていません。

実動モードで TCP/IP for z/VSE を実行する前に、永続製品キーを供給する必要があります。この製品キーは、契約したライセンスに基づきます。「構成」データに割り振ったサブライブラリー (例えば、PRD2.CONFIG) に実働の製品キーを保管し、このサブライブラリーを LIBDEF 検索順序の先頭に置くことをお勧めします。このようにすると、保守の適用や製品の再インストールによって、実動キーがオーバーレイされることがありません。

製品の使用可能化は、2 つのフェーズで行われますが、それらのフェーズは、下記の例に示すようにジョブ・ストリームを使用して生成できます。

GPS (汎用プリント・サーバー) を使用する計画がある場合、Application Pak 用のキーに加えて、GPS 用の別のキーをインストールする必要があります。

製品キーのインストール

```
// JOB KEY
// LIBDEF *,SEARCH=PRD2.TCPIPC
// LIBDEF PHASE,CATALOG=PRD2.CONFIG
// OPTION CATAL
// EXEC ASMA90,SIZE=(ASMA90,50K)
         PRODKEY  1234-5678-9012-3456-7890 /* APPLICATION PAK */
         PRODKEY  3456-7890-1234-5678-9012 /* GPS */
         END
/*
// EXEC LNKEDT
/&
```

カスタマー情報の定義

```
// JOB TCPCUS
// LIBDEF *,SEARCH=(PRD2.TCPIPC)
// LIBDEF PHASE,CATALOG=PRD2.CONFIG
// OPTION CATAL
// EXEC ASSEMBLY
         CUSTDEF DEFINE,                                              X
               NAME='IBM z/VSE Development',                          X
               NUMBER=C123-456-7890
         END
/*
// EXEC LNKEDT
/&
```

注:

1. 上記の例で、PRD2.CONFIG は、TCP/IP for z/VSE の構成データのインストール先となるライブラリーの名前です。
2. ソフトウェアのライセンス契約の完了後、例に示された文字列を実際の製品キーと置き換えることができます。この例で表示するキーは、例示のみを目的としています。
3. TCP/IP for z/VSE によって使用されるカスタマー番号 (上の 2 番目の例にあります) は、IBM カスタマー番号ではありません。CUSTDEF マクロが使用するカスタマー番号は、製品キーが指定されているのと同じメモに記載されています。

マイグレーション考慮事項

z/VSE にプリインストールされている TCP/IP for z/VSE は、VM/VSE 環境で、TCP/IP for z/OS® および TCP/IP for z/VM とともに使用できます。どちらの製品も、イントラネットやインターネットへの一般的なゲートウェイとして使用できます。これらの製品を組み合わせて使用するために必要な構成については、TCP/IP の資料を確認してください。例えば、CTCA 接続を使用することができます。あるいは、TCP/IP for z/VSE は、非 VSE システム上の任意の TCP/IP 製品に接続するために使用することもできます (この TCP/IP インプリメンテーションが TCP/IP 標準に準拠している場合に限ります)。

IBM から TCP/IP for z/VSE を購入することを選択し、それを z/VSE システム上の IBM でも Connectivity Systems でもない別の TCP/IP インプリメンテーションと共に使用する予定の場合、正式にテストされていない環境で実行することになります。このような場合、問題が起こらないように両製品を注意深く構成する必要があります。例えば、それらの製品は同じファイル名を使用する可能性があり、その場合に LIBDEF チェーンが適切にセットアップされていなければ、製品の動作は予測不能になります。これは例えば、duplicate SOCKET.H C 言語ヘッダー・ファイルが重複しているなどの場合です。

z/VSE と共にプリインストールされた TCP/IP 以外の製品を実行すると決めた場合は、IBM 提供の delete ジョブ (ICCF ライブラリー 59 内のスケルトン DELTCPIP を参照) を実行して、この TCP/IP が、プリインストールされた TCP/IP for z/VSE に干渉しないようにしてください。

Connectivity Systems 社のものではない他の TCP/IP 製品から TCP/IP for z/VSE にマイグレーションする場合、製品と共に提供される構成ステップに従ってください。その際、ホスト IP アドレスなど現行の TCP/IP 固有パラメーターを使用すれば、製品のセットアップが簡単になります。

z/VSE 上で TCP/IP for z/VSE にマイグレーションする前に、Connectivity Systems Inc. または同社のディストリビューターの 1 つから提供される TCP/IP for z/VSE を使用していた場合、以下の点を考慮してください。

- z/VSE は、TCP/IP for z/VSE を PRD2.TCPIPC システム・ライブラリーにプリインストールします。Connectivity Systems Inc. のインストール推奨事項に従って TCP/IP for z/VSE を専用サブライブラリーにインストールしていた場合、そのサブライブラリーをデフォルト LIBDEF チェーンから除去してください。
- TCP/IP for z/VSE は、PRD2.TCPIPC に格納されます。これ以外の TCP/IP サブライブラリーを参照するジョブは、変更する必要があります。これには、TCP/IP スタートアップ・ジョブ自体に加えて、バッチ・ジョブ内から LPR、FTP、または TELNET セッションなどを実行するジョブもすべて含まれます。TCP/IP 関連の開発を独自に行っている場合、それぞれの開発手順に影響する可能性があります。
- Connectivity Systems Inc. が推奨するように TCP/IP 固有の構成ファイル (IPINITxx.L や NETWORK.L など) を別のサブライブラリーに保管していなかった場合は、変更されたそれらのファイルを PRD2.CONFIG に移動し、次回の z/VSE サービス・リフレッシュでそれらのファイルが置き換えられないようにする必要があります。これには、IPXLATE 変換フェーズや、提供された TCPAPPL ソース・ブック内の Telnet 関連の端末定義に対して行った、機能強化/修正または固有の置換が含まれます。
- Connectivity Systems Inc. からの製品キーを用いてアセンブルした PRODKEYS フェーズを名前変更し、新しい PRODKEYS フェーズを、IBM から提供された製品キーを使用して生成してください。

  Connectivity Systems Inc. からの製品キーでは PRODKEYS フェーズの生成のみが必要ですが、IBM 提供のキーは、 6 ページの『カスタマー情報の定義』に説明されているように、それに加えて CUSTDEF フェーズの生成を必要とします。IBM 提供のキーの妥当性検査では、LE/VSE C ランタイム環境がアクセス可能であることが必要です。このために TCP/IP のスタートアップ・ジョブの LIBDEF 定義に PRD2.SCEEBASE を組み込んでください。
- TCP/IP が定義した仮想ファイル・システムは、z/VSE へのマイグレーションによって変更される場合があります。必要に応じて IPINITxx.L 構成メンバーを更新してください。
- z/VSE と共に提供される TCP/IP for z/VSE は、MSHP で完全に制御されます。つまり、以前の製品セットアップでは行ったかもしれませんが、Connectivity Systems Inc. 提供のサービス・パックをシステムに適用してはなりません。その代わりに、IBM 提供の PTF のみをインストールしてください。そうしないと、サポートされない環境で実行することになってしまいます。PTF 以外の修正を適用すると、システムの品質が低下したり、予測不能の影響が発生したりする可能性があります。

- TCP/IP for z/VSE アプリケーションを独自に開発した場合、以下の点に注意してください。
  - TCP/IP for z/VSE **SOCKET** マクロを使用していたアセンブラー・アプリケーションがある場合、それらの再アセンブルが必要な場合があります。このマクロには、IBM の TCP/IP for z/VSE でリフレッシュされた可能性のあるインライン・コードが含まれています。
  - 製品と共に提供された BSD-C ソケット・インターフェースを使用していたアプリケーションがある場合、また、COBOL、PL/I、またはアセンブラーのプログラム内の EXEC TCP ソース・ステートメントを解決するために製品のプリプロセッサーを使用していた場合は、それらの再リンクが必要な場合があります。それらのアプリケーションにリンクされた IPNxxxx.OBJ ファイルが保守された場合に、これが必要になることがあります。

TCP/IP for z/VSE のファイアウォール

TCP/IP for z/VSE には、基本的なファイアウォール・セキュリティー機能があります。

ほとんどの VSE インストールでは、VSE システムへのアクセスを許可されている既知の IP アドレスの数が限られています。したがって、z/VSE のファイアウォール・シールドは、ホワイトリスト・ベースのファイアウォール実装です。これはデフォルトでのセキュリティーと考えることもできます。つまり、IP アドレスが VSE システムとの通信を特に許可されていない限り、アクセスは拒否されるということです。

活動化

ファイアウォールが初期化される前にリモート・システムがファイアウォール・シールドを迂回してしまう可能性を避けるために、このシールドは、TCP/IP 始動時の早い段階でロードされます。ファイアウォールを活動化するメカニズムには、使用されるカスタム構成フェーズを識別するパラメーター **FIREWARN=** または **FIREWALL=** が用いられます。この場合、TCP/IP の始動は、以下のいずれかと類似しています。

WARN モードで活動化する場合:

```
// EXEC IPNET,SIZE=IPNET,PARM='ID=00,INIT=IPINIT00,FIREWARN=FIREWALL'
```

FAIL モードで活動化する場合:

```
// EXEC IPNET,SIZE=IPNET,PARM='ID=00,INIT=IPINIT00,FIREWALL=FIREWALL'
```

どちらのコマンドも、許可される IP アドレスのリストが含まれる、構成可能なファイアウォール・フェーズをロードします。構成フェーズのデフォルト名は FIREWALL で、これは複数のスタック間で共有できます。複数のスタックを実行する場合は、異なる設定ができるように、固有のフェーズ名を作成することもできます。 IP アドレスのそれぞれの範囲では、許可される VSE ポート (TCP または UDP)、およびリモート IP アドレス範囲からの ICMP (ping) の許可の可否を指定できます。このフェーズが正常にロードされた場合、ファイアウォール・シールドは指定されたモードで活動化されます。

ファイアウォールが FAIL モードで活動化された場合、ファイアウォール構成フェーズで定義されていないリモート IP アドレスから受け取ったデータグラムは破棄され、違反メッセージが発行されます。

ファイアウォールが WARN モードで活動化された場合、同じ違反メッセージが発行されますが、関連するデータグラムは破棄されません。 WARN モードは、ファイアウォールの初期セットアップで、リモート IP アドレスがアクセスを試行する VSE 上のポート番号を識別するために役立ちます。

ファイアウォール構成、コマンド、およびメッセージについて詳しくは、TCP/IP for z/VSE の資料を参照してください。

第 2 章 TCP/IP for z/VSE の構成

TCP/IP for z/VSE のインストール方法

TCP/IP for z/VSE は、z/VSE と共に PRD2.TCPIPC ライブラリーにプリインストールされます。個別設定情報、例えば、キー定義、カスタマー定義、TCP/IP スタートアップ・メンバーなどは、PRD2.TCPIPC 内に保持しないでください。一部のモジュールに対して PTF の適用あるいはシステム・リフレッシュとしての高速サービス・アップグレード (FSU) により保守が行われる可能性があるため、別の場所を使用する必要があります。

TCP/IP for z/VSE パーティションのスタートアップ

z/VSE は、デフォルトのパーティション F7 を TCP/IP for z/VSE に対して定義します。デフォルト・パーティション・スタートアップ・メンバー TCPSTART.Z が PRD2.TCPIPC 内にあります。ご使用の構成に合わせてこれを調整し、DTRIINIT ユーティリティーを使用して VSE/POWER RDR 待ち行列に入れることができます (下記の例を参照してください)。更新後のメンバーは PRD2.CONFIG に保管できます。

TCP/IP のデフォルトのパーティションは F7 で、デフォルトで 20 MB です。本製品が 31 ビットを活用していることによるメリットを得るために、少なくとも 30 MB のパーティションで TCP/IP for z/VSE を使用することを強くお勧めします。

TCP/IP は、優先順位の高い VSE パーティションを必要とすることに注意してください。従って、タイミングに依存する VTAM® などの他の製品と同様に、VTAM とほぼ同じ PRTY を持つパーティションを使用することを強くお勧めします。例えば、TCP/IP が CICS にサービスを提供して Telnet または MQSeries® の使用しなければならない場合には、特にこれを推奨します。

例

以下のジョブ・ストリームを使用して、TCP/IP for z/VSE スタートアップ・メンバー TCPSTART.Z を POWER® RDR 待ち行列にロードできます。

```
* $$ JOB JNM=TCPLOAD,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB TCPLOAD LOAD TCPIP STARTUP INTO POWER
// LIBDEF *,SEARCH=IJSYSRS.SYSLIB
// EXEC DTRIINIT
ACCESS PRD2.TCPIPC
LOAD TCPSTART.Z
/*
/&
* $$ EOJ
```

TCPSTART.Z は、次のようなものです。

```
* $$ JOB JNM=TCPSTART,CLASS=7,DISP=K
* $$ LST CLASS=A,DISP=D
// JOB TCPIP
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD2.TCPIPC,PRD2.SCEEBASE)
```

```
// EXEC PROC=DTRICCF
// SETPFIX LIMIT=(400K,2100K)
// EXEC IPNET,SIZE=IPNET,PARM='ID=00,INIT=IPINIT00',DSPACE=2M
/&
* $$ EOJ
```

注:

1. 例で、PRD2.TCPIPC は TCP/IP for z/VSE がプリインストールされたライブラリーであり、PRD2.CONFIG はご使用のシステムに依存する値 (初期設定メンバーおよび許可コード) を置いたライブラリーです。
2. PRD2.SCEEBASE ライブラリーは、LE/VSE C ランタイム環境を含み、IBM 製品キー検証に必要です。
3. LIBDEF ステートメントに「*」を指定するようにしてください。「フェーズ」を指定すると、TCP/IP for z/VSE は初期設定メンバー IPINIT00.L を見つけられなくなります。
4. システム提供の TCP/IP スタートアップ・ジョブを使用しない場合は、PRD2.SCEEBASE に入っている LE/VSE C ランタイムを LIBDEF 定義に含めてください。これは、IBM 製品キーの妥当性検査を適正に行うのに必須です。

CICS の構成

TCP/IP for z/VSE には、いくつかの CICS ベースのクライアントが組み込まれています。これらのクライアントによって、CICS ユーザーは TCP/IP を使用して例えば以下の操作を実行できます。

- Telnet を介して他のプラットフォームおよびアプリケーションに (CICS から) ログオンする。例えば、ユーザーは CICS から UNIX システムにログオンできます。
- TCP/IP for z/VSE FTP サーバーとリモート FTP サーバー間のファイル転送を開始する。
- LPR を使用してファイルを印刷する。
- Ping クライアントを使用してネットワーク接続をチェックする。

CICS のセットアップ

- CICS パーティションの検索チェーン中に必ず TCP/IP for z/VSE が設定されているようにしてください。これは、次のように CICS スタートアップ JCL を変更することによって行うことができます。

  // **LIBDEF *,SEARCH=**(*lib*,*lib*,**PRD2.TCPIPC**)
- TCP/IP と共に使用される CICS に、プログラムおよびトランザクションを定義してください。

注:

1. グループ「TCPIP」およびリスト「VSELIST」を使用するかどうかは任意です。サイトの要件に合わせて調整を行ってもかまいません。
2. DFHPPTIP.A は、TCP/IP for z/VSE 製品と共に出荷され、以下のような内容です。

   DFHPPTIP — CICS 処理プログラム・テーブル

```
PPTIP    TITLE    'DFHPPTIP - Cics Processing Program Table'
         DFHPPT   TYPE=INITIAL,                                        *
               SUFFIX=IP
         DFHPPT   TYPE=ENTRY,              Entry                       *
               PROGRAM=TELNET01,           Program Identification     *
               RSL=PUBLIC,                 Public Program              *
               PGMLANG=ASSEMBLER           Assembler
         DFHPPT   TYPE=ENTRY,              Entry                       *
               PROGRAM=FTP01,              Program Identification     *
               RSL=PUBLIC,                 Public Program              *
               PGMLANG=ASSEMBLER           Assembler
         DFHPPT   TYPE=ENTRY,              Entry                       *
               PROGRAM=CLIENT01,           Program Identification     *
               RSL=PUBLIC,                 Public Program              *
               PGMLANG=ASSEMBLER           Assembler
         DFHPPT   TYPE=FINAL
         END
```

3. DFHPCTIP.A は、TCP/IP for z/VSE 製品と共に出荷され、以下のような内容です。

   DFHPCTIP — CICS トランザクション・テーブル

```
PCTIP    TITLE  'DFHPCTIP - Cics Transaction Table'
         DFHPCT   TYPE=INITIAL,                                        *
               SUFFIX=IP
         DFHPCT   TYPE=ENTRY,              Entry                       *
               TRANSID=TELN,               Transaction Name            *
               PROGRAM=TELNET01,           Program Identification     *
               RSL=PUBLIC                  Public
         DFHPCT   TYPE=ENTRY,              Entry                       *
               TRANSID=teln,               Transaction Name            *
               PROGRAM=TELNET01,           Program Identification     *
               RSL=PUBLIC                  Public
         DFHPCT   TYPE=ENTRY,              Entry                       *
               TRANSID=TELC,               Transaction Name            *
               PROGRAM=TELNET01,           Program Identification     *
               RSL=PUBLIC                  Public
         DFHPCT   TYPE=ENTRY,              Entry                       *
               TRANSID=TELW,               Transaction Name            *
               PROGRAM=TELNET01,           Program Identification     *
               RSL=PUBLIC                  Public
         DFHPCT   TYPE=ENTRY,              Entry                       *
               TRANSID=TELR,               Transaction Name            *
               PROGRAM=TELNET01,           Program Identification     *
               RSL=PUBLIC                  Public
         DFHPCT   TYPE=ENTRY,              Entry                       *
               TRANSID=FTP,                Transaction Name            *
               PROGRAM=FTP01,              Program Identification     *
               RSL=PUBLIC                  Public
         DFHPCT   TYPE=ENTRY,              Entry                       *
               TRANSID=ftp,                Transaction Name            *
               PROGRAM=FTP01,              Program Identification     *
               RSL=PUBLIC                  Public
         DFHPCT   TYPE=ENTRY,              Entry                       *
               TRANSID=FTPC,               Transaction Name            *
               PROGRAM=FTP01,              Program Identification     *
               RSL=PUBLIC                  Public
         DFHPCT   TYPE=ENTRY,              Entry                       *
               TRANSID=FTPW,               Transaction Name            *
               PROGRAM=FTP01,              Program Identification     *
               RSL=PUBLIC                  Public
         DFHPCT   TYPE=ENTRY,              Entry                       *
               TRANSID=FTPR,               Transaction Name            *
               PROGRAM=FTP01,              Program Identification     *
               RSL=PUBLIC                  Public
         DFHPCT   TYPE=ENTRY,              Entry                       *
```

```
      TRANSID=LPR,              Transaction Name           *
      PROGRAM=CLIENT01,         Program Identification     *
      RSL=PUBLIC                Public
DFHPCT   TYPE=ENTRY,            Entry                       *
      TRANSID=lpr,              Transaction Name           *
      PROGRAM=CLIENT01,         Program Identification     *
      RSL=PUBLIC                Public
DFHPCT   TYPE=ENTRY,            Entry                       *
      TRANSID=PING,             Transaction Name           *
      PROGRAM=CLIENT01,         Program Identification     *
      RSL=PUBLIC                Public
DFHPCT   TYPE=ENTRY,            Entry                       *
      TRANSID=ping,             Transaction Name           *
      PROGRAM=CLIENT01,         Program Identification     *
      RSL=PUBLIC                Public
DFHPCT   TYPE=ENTRY,            Entry                       *
      TRANSID=TCPC,             Transaction Name           *
      PROGRAM=CLIENT01,         Program Identification     *
      RSL=PUBLIC                Public
DFHPCT   TYPE=ENTRY,            Entry                       *
      TRANSID=TCPW,             Transaction Name           *
      PROGRAM=CLIENT01,         Program Identification     *
      RSL=PUBLIC                Public
DFHPCT   TYPE=ENTRY,            Entry                       *
      TRANSID=TCPR,             Transaction Name           *
      PROGRAM=CLIENT01,         Program Identification     *
      RSL=PUBLIC                Public
DFHPCT   TYPE=FINAL
END
```

CICS/TS 2.1 の場合の例

メンバー IPNCSD.Z を使用して、プログラムおよびトランザクションを CICS に定義します。これに加えて、 IPNCSD.Z を使用するためのメンバー IPNCSDUP.Z があります。以下に示すのは、IPNCSDUP.Z の例です。

```
* $$ JOB JNM=IPNCSDUP,CLASS=0,DISP=D
// JOB IPNCSDUP
* SHUT DOWN CICS FIRST
// PAUSE CLOSE DFHCSD FILE IF CICS IS UP : CEMT SE FI(DFHCSD) CLOSE
/*
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD1.BASE,PRD2.SCEEBASE)
// EXEC DFHCSDUP,SIZE=600K        INIT AND LOAD CICS
  DELETE GROUP(TCPIP)
* $$ SLI MEM=IPNCSD.Z,S=(PRD1.BASE)
  ADD GROUP(TCPIP) LIST(VSELIST)
  LIST ALL
/*
/&
* $$ EOJ
```

注:

1. グループ「TCPIP」およびリスト「VSELIST」を使用するかどうかは任意です。サイトの要件に合わせて調整を行ってもかまいません。
2. IPNCSD.Z は、TCP/IP for z/VSE 製品と共に出荷され、以下のような内容です。

   TCP/IP for z/VSE と一緒に出荷される IPNCSD.Z

```
*-------------------------------------------------------------------*
*    FOLLOWING ARE THE PPT ENTRIES REQUIRED FOR TCP/IP for z/VSE    *
*-------------------------------------------------------------------*
  DEFINE  PROGRAM(TELNET01) GROUP(TCPIP)
```

```
         LANGUAGE(ASSEMBLER)
  DEFINE  PROGRAM(FTP01)    GROUP(TCPIP)
         LANGUAGE(ASSEMBLER)
  DEFINE  PROGRAM(CLIENT01) GROUP(TCPIP)
         LANGUAGE(ASSEMBLER)
*-------------------------------------------------------------------*
*    FOLLOWING ARE THE PCT ENTRIES REQUIRED FOR TCP/IP for z/VSE    *
*-------------------------------------------------------------------*
  DEFINE TRANSACTION(TRAC)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(trac)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(REXE)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(rexe)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(DISC)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(disc)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(EMAI)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(emai)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(PING)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(ping)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(TELN)   GROUP(TCPIP)
           PROGRAM(TELNET01)
  DEFINE TRANSACTION(teln)   GROUP(TCPIP)
           PROGRAM(TELNET01)
  DEFINE TRANSACTION(TELC)   GROUP(TCPIP)
           PROGRAM(TELNET01)
  DEFINE TRANSACTION(TELW)   GROUP(TCPIP)
           PROGRAM(TELNET01)
  DEFINE TRANSACTION(TELR)   GROUP(TCPIP)
           PROGRAM(TELNET01)
  DEFINE TRANSACTION(FTP)    GROUP(TCPIP)
           PROGRAM(FTP01)
  DEFINE TRANSACTION(ftp)    GROUP(TCPIP)
           PROGRAM(FTP01)
  DEFINE TRANSACTION(FTPC)   GROUP(TCPIP)
           PROGRAM(FTP01)
  DEFINE TRANSACTION(FTPW)   GROUP(TCPIP)
           PROGRAM(FTP01)
  DEFINE TRANSACTION(FTPR)   GROUP(TCPIP)
           PROGRAM(FTP01)
  DEFINE TRANSACTION(TCPC)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(TCPW)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(TCPR)   GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(LPR)    GROUP(TCPIP)
           PROGRAM(CLIENT01)
  DEFINE TRANSACTION(lpr)    GROUP(TCPIP)
           PROGRAM(CLIENT01)
*-------------------------------------------------------------------*
*                 END OF TCP/IP MEMBER                              *
*-------------------------------------------------------------------*
```

HTMLINST.Z

HTML (Hyper Text Markup Language) 文書の交換用に、HTTP が使用されます。HTML 文書は、表示可能テキストと、文書がどのように表示されるのかを指示する HTML タグが混在して含まれているファイルです。他の文書へのリンク、グラフィックスの組み込み、または特殊な効果を実現できる HTML 要素もあります。

TCP/IP for z/VSE では、セキュリティー上の理由から、以下の特別な HTML ファイルが提供されています。

- PASSWORD.HTML
- VIOLATED.HTML
- BLANKING.HTML

PRD2.TCPIPC 内のメンバー HTMLINST.Z には、これらの特別な HTML ファイルのデフォルトのメンバーを生成するジョブ・ストリームが含まれています。DTRIINIT ユーティリティーを使用して、メンバー HTMLINST.Z を VSE/POWER RDR 待ち行列にロードできます。

例

```
* $$ JOB JNM=HTMLLOAD,CLASS=A,DISP=D
* $$ LST CLASS=A,DISP=D
// JOB HTMLLOAD LOAD HTMLINST.Z INTO POWER
// LIBDEF *,SEARCH=IJSYSRS.SYSLIB
// EXEC DTRIINIT
ACCESS PRD2.TCPIPC
LOAD HTMLINST.Z
/*
/&
* $$ EOJ
```

個々の HTML メンバーの使用方法についての詳しい説明は、「*TCP/IP for VSE Installation Guide*」の『Configuring the HTTP Daemon (HTTP デーモンの構成)』章の『Security (セキュリティー)』セクションにあります。

第 3 章 TCP/IP for z/VSE 構成ダイアログ

TCP/IP for z/VSE に関するシステムの構成を始める前に、 3 ページの『第 1 章 概要』をお読みください。

TCP/IP for z/VSE は、構成作業を完了してからでないと使用できません。

この作業は、必要な定義を関連 TCP/IP for z/VSE ライブラリー・メンバーおよび定義ジョブに設定することによって、手動で実行できます。作業を支援するために、いくつかの構成メンバーが用意されています。例えば、TCPSTART.Z (詳しくは、 11 ページの『TCP/IP for z/VSE パーティションのスタートアップ』を参照)、TCPAPP00.B (Telnet デーモン用 VTAM 定義のサンプル)、IPINIT00.L (TCP/IP for z/VSE 初期設定メンバーのサンプル) です。あるいは、手動での作業ではなく、 18 ページの『IUI ベースの構成ダイアログを使用した TCP/IP の構成』を使用することもできます。

TCP/IP for z/VSE を実動モードで実行する前に、 4 ページの『製品キーの提供』に説明されているとおり、製品キーをインストールする必要があります。

構成ダイアログを使用した TCP/IP の構成

TCP/IP for z/VSE を開始する前に、構成情報を指定しなければなりません。以下を指定できます。

一般情報
: いくつかの一般的な構成情報が必要です。例えば、TCP/IP for z/VSE のホスト IP アドレスを指定する必要があります。

リンク
: TCP/IP for z/VSE が外部通信のために使用する、各デバイス、コントローラー、および接続機構を指示する必要があります。

デーモン
: サービス・デーモンの定義を指定する必要があります。デーモンは、エンド・ユーザーにサービスを提供するルーチンです。例えば、FTP は、VSE ファイル・システムへのアクセスを提供するサービス・デーモンです。

ルーティング情報
: ご使用の構成によっては、TCP/IP for z/VSE 製品にルーティング情報を定義する必要があります。この情報は、他の TCP/IP プラットフォームとの接続を制御するために使用されます。

実行方法

すべての構成情報は、一連のコンソール・オペレーター・コマンドによって指定されます。したがって、単に TCP/IP for z/VSE 製品を開始してから、コマンドによってすべての構成データを入力するだけで済みます。またはさらに簡便な方法とし

て、一連の構成コマンドを、初期化ライブラリー・メンバー IPINITxx.L に入れることもできます。この方法は、「*TCP/IP for VSE Installation Guide*」で詳しく説明されています。

最も簡単な方法として、『IUI ベースの構成ダイアログを使用した TCP/IP の構成』に説明されているように IUI ベースの構成ダイアログを使用することもできます。

デフォルトのメンバー IPINIT00.L が z/VSE と共に VSE ライブラリー PRD2.TCPIPC に入って出荷されます。これには、デフォルト値に設定された多数の構成パラメーターが含まれています。独自の構成を開発するための開始点としてこれを使用すると便利です。

独自の初期設定メンバーは、構成データ用に確保したサブライブラリー (例えば、PRD2.CONFIG) に保管してください。このようにすると、TCP/IP 製品または z/VSE システムへの保守の適用中に、そのメンバーが誤って置き換えられることがなくなります。

IUI ベースの構成ダイアログを使用した TCP/IP の構成

「**TCP/IP Configuration (TCP/IP** 構成**)**」パネル (ファスト・パス **245**) で対話式インターフェースが拡張され、TCP/IP 環境の構成がより簡単にできるようになりました。下に示したパネルで TCP/IP パラメーターを定義し終わったら、PF5 (PROCESS) を押して、ジョブ・ストリームを作成する必要があります。このジョブ・ストリームが、システム・ライブラリー PRD2.CONFIG 内の関連する構成メンバー (IPINIT00.L など) を更新します。PRD2.TCPIPC 内にメンバーが存在しない場合、PRD1.BASE 内のシステム・デフォルト・メンバー IPINIT00 が使用されます。以下に、「***TCP/IP Configuration (TCP/IP*** 構成**)**」パネル (19 ページの図 1) を示し、このパネルに表示されるパラメーターの値を定義できる一連のパネルを順に説明します。必要な値を入力し終わったら、このパネルで PF5 (PROCESS) を押します。

TELNET デーモンが追加された場合、PF9=VTAM を押してメンバー TCPAPP00.B を更新してください (詳しくは、 23 ページの図 10を参照)。

```
CON$SEL                    TCP/IP CONFIGURATION

Enter the required data and press ENTER.


To modify one or more of the following TCP/IP parameters,
place a  1 next to it.

      _     SET         Modify SET IPADDR or SET MASK command
      _     LINK        Modify DEFINE LINK command
      _     ADAPTER     Modify DEFINE ADAPTER command
      _     ROUTE       Modify DEFINE ROUTE command
      _     TELNETD     Modify DEFINE TELNETD command









PF1=HELP       2=REDISPLAY  3=END                    5=PROCESS
                            9=VTAM
```

図 *1.* 「*TCP/IP Configuration (TCP/IP* 構成*)*」パネル *CON$SEL*

SET を選択すると、「***TCP/IP Configuration: Set IPADDR and MASK (TCP/IP*** 構成***: IPADDR*** および ***MASK*** の設定***)***」パネル CON$SIP が表示されます。このパネルには、SET IPADDR および SET MASK ステートメント、およびこの z/VSE クライアントの DEFINE NAME ステートメントに対して既に定義された値が表示されます。これらの値は変更可能です。

```
CONP$SIP           TCP/IP CONFIGURATION: SET IPADDR AND MASK

Enter the required data and press ENTER.


Specify the parameters for the SET IPADDR and the SET MASK command.

IPADDR.....      ___  ___  ___  ___   default network address

MASK.......      ___  ___  ___  ___   value of the mask

NAME.......      ____________________  this z/VSE client









PF1=HELP       2=REDISPLAY  3=END
```

図 *2.* 「*TCP/IP Configuration : Set IPADDR and MASK (TCP/IP* 構成*: IPADDR* および *MASK* の設定*)*」パネル

「***TCP/IP Configuration (TCP/IP*** 構成***)***」パネル (図 1) で LINK を選択すると、すべての定義済みリンクのリストが「***TCP/IP Configuration: Link List (TCP/IP*** 構成***:*** リンク・リスト***)***」パネルに表示されます。

```
CON$LNKS              TCP/IP CONFIGURATION: LINK LIST

Enter the required data and press ENTER.

OPTIONS: 1 = ADD LINK    2 = ALTER LINK   3 = ADD ROUTE
         4 = ADD ADAPTER 5 = DELETE LINK

 OPT       LINKID           DEVICE  TYPE    DATAPATH  IPADDR          PORT
  *        OSAFE            500     OSAX    502                       F14FEL1
  -
  -
  -
  -
  -
  -
  -
  -
  -

PF1=HELP       2=REDISPLAY  3=END                    5=PROCESS
```

図 *3.* 「*TCP/IP Configuration: Link List (TCP/IP* 構成*:* リンク・リスト*)*」パネル

オプション 1 の ADD LINK を入力すると、以下のパネルが表示されます。

```
CON$LNK                 TCP/IP CONFIGURATION: LINK

Enter the required data and press ENTER.


LINK ID..... _______________    Enter the unique name of the link.

TYPE..................... _______    Specify the link type.

DEVICE................... ___        Enter the unit address at which the
                                     network connection device resides.

DATAPATH................. ___        Enter the unit address of the data-
                                     path.

IPADDR.....      ___  ___  ___  ___  Associated IP address.

OSAPORT.................. _________  Specify the port name or OSAPORT.


PF1=HELP       2=REDISPLAY  3=END
```

図 *4.* 「*TCP/IP Configuration: Link (TCP/IP* 構成*:* リンク*)*」パネル

入力が正しい場合、ダイアログは「***TCP/IP Configuration: LINK LIST (TCP/IP*** 構成***:*** リンク・リスト***)***」パネル (図 3) に戻ります。

「***TCP/IP Configuration (TCP/IP*** 構成***)***」パネル (19 ページの図 1) で ADAPTER を選択すると、以下のパネルが表示されます (このパネルは、タイプ OSA または 3172 のリンクについてのみ可能であることに注意してください)。

```
CON$APTS            TCP/IP CONFIGURATION: ADAPTER LIST

Enter the required data and press ENTER.

OPTIONS: 1 = ADD ADAPTER 2 = ALTER ADAPTER
         5 = DELETE ADAPTER

 OPT       LINKID               TYPE    NUMBER
  _        ELKEL                FDDI    01
  _        F234567890123456     FODI    02
  _
  _


PF1=HELP       2=REDISPLAY  3=END                    5=PROCESS
```

図 *5.* 「*TCP/IP Configuration: Adapter List (TCP/IP* 構成*:* アダプター・リスト*)*」パネル

オプション 1 の ADD ADAPTER を入力すると、以下のパネルが表示されます。

```
CON$APT                TCP/IP CONFIGURATION: ADAPTER

 Enter the required data and press ENTER.


 LINK ID..... ________________    Enter the link id

 TYPE..................... ________    Specify the type of adapter.

 NUMBER..................... __          Enter the adapter number.









 PF1=HELP       2=REDISPLAY  3=END
```

図 *6.* 「*TCP/IP Configuration : Adapter (TCP/IP* 構成*:* アダプター*)*」パネル

入力が正しい場合、ダイアログは「***TCP/IP Configuration (TCP/IP*** 構成***): CON$SEL***」パネル (19 ページの図 1) に戻ります。

この時点で、「***TCP/IP Configuration (TCP/IP*** 構成***)***」パネル (19 ページの図 1) で経路を選択することができ、そうすると以下のパネルが表示されます。

```
CON$RTS              TCP/IP CONFIGURATION: ROUTE LIST

Enter the required data and press ENTER.

OPTIONS: 1 = ADD ROUTE
         5 = DELETE ROUTE

 OPT  ROUTEID           LINKID             IPADDR           GATEWAY
 _    ALL               VM_TCPIP           0.0.0.0          9.164.186.5
 _    R234567890123456  L234567890123456   155.155.155.155  111.222.33.0
 _    R2                ELKEL2             9.9.9.9          121.231.34.0
 _
 _
 _
 _
 _
 _
 _

PF1=HELP       2=REDISPLAY  3=END                    5=PROCESS
```

図 *7.* 「*TCP/IP Configuration: Route List (TCP/IP* 構成*:* 経路リスト*)*」パネル

オプション 1=ADD ROUTE を入力すると、「DEFINE ROUTE (経路の定義)」パネルが表示されます。「***TCP/IP Configuration: LINK LIST (TCP/IP*** 構成*:*** リンク・リスト**)**」パネル (20 ページの図 3) で 3=ADD ROUTE を入力しても、同じパネルを表示できます。その場合には、LINKID が既に指定された状態になっています。

```
CON$ROUT             TCP/IP CONFIGURATION: DEFINE ROUTE

Enter the required data and press ENTER.




ROUTE ID..... ________________   Unique name of the route.

LINK ID..... ________________    Name of the associated link.

IPADDR.....     ___  ___  ___  ___   Associated IP address.

GATEWAY....     ___  ___  ___  ___   Full network address of a gateway




PF1=HELP       2=REDISPLAY  3=END
```

図 *8.* 「*TCP/IP Configuration: Define Route (TCP/IP* 構成*:* 経路の定義*)*」パネル

「***TCP/IP Configuration (TCP/IP*** 構成**)**」パネル (19 ページの図 1) で TELNET DAEMON を選択すると、「TELNET LIST (Telnet リスト)」パネルが表示されます。

```
CON$TELS             TCP/IP CONFIGURATION: TELNET LIST

Enter the required data and press ENTER.

OPTIONS: 1 = ADD TELNET DAEMON   2 = ALTER TELNET DAEMON
         5 = DELETE DAEMON

 OPT       DAEMONID           TARGET    TERMNAME  COUNT   LOGMODE
  _        MYTEL              DBDCCICS  T1000     20      U
  _
  _
  _
  _
  _
  _
  _
  _
  _

PF1=HELP       2=REDISPLAY  3=END                    5=PROCESS
```

図 *9.* 「*TCP/IP Configuration: TELNET LIST (TCP/IP* 構成*: Telnet* リスト*)*」パネル

オプション 1=ADD TELNET DAEMON を入力すると、以下のパネルが表示されます。

```
CON$TELD             TCP/IP CONFIGURATION: TELNET DAEMON

 Enter the required data and press ENTER.

 DAEMON ID..... ________________    Enter the unique name of the daemon

 TARGET...................... ________    Enter the name of the VTAM applica-
                                          tion id you are connecting to

 TERMNAME.................... ________    Enter the VTAM LU name assigned to
                                          the remote terminal.

 COUNT....................... __          Count for multiple telnet daemons.

 LOGMODE..................... _           VTAM LOGMODEs for the LU session.


 PF1=HELP       2=REDISPLAY  3=END
```

図 *10.* 「*TCP/IP Configuration: TELNET DAEMON (TCP/IP* 構成*: Telnet* デーモン*)*」パネル

TELNET デーモンを追加するときには、TERMname が VTAM アプリケーションとして定義されていなければなりません。従って、ブック TCPAPP00 が VTAM 構成メンバー ATCCON00.B に含まれています。アプリケーション定義を TCPAPP00.B に自動的に追加する機能がダイアログに用意されています。作成は *PF9=VTAM* を押すことでトリガーされます。更新を有効にするには、「***TCP/IP Configuration (TCP/IP*** 構成*)***」パネル (19 ページの図 1) で PF5=PROCESS を押す必要があります。そうすると、PRD2.CONFIG 内の更新された TCP/IP スタートアップ・メンバー IPINIT00.L と、要求された場合は VTAM ブック TCPAPP00.B とをカタログするジョブが作成されます。

第 4 章 TCP/IP for z/VSE によるセキュリティー・マネージャーの活用

このセクションでは、TCP/IP for z/VSE がどのように基本セキュリティー・マネージャー (BSM) の機能を利用するのかを説明します。このインプリメンテーションが該当するのは、z/VSE です。

TCP/IP セキュリティー検査のための BSM 機能の使用

TCP/IP によってさまざまなプラットフォームが VSE と通信できるため、それに応じてセキュリティー要件が発生します。これが、TCP/IP for z/VSE が VSE リソースを保護するための多数の機能を備えている理由です。詳しくは、「*TCP/IP for VSE Command Reference*」を参照してください。

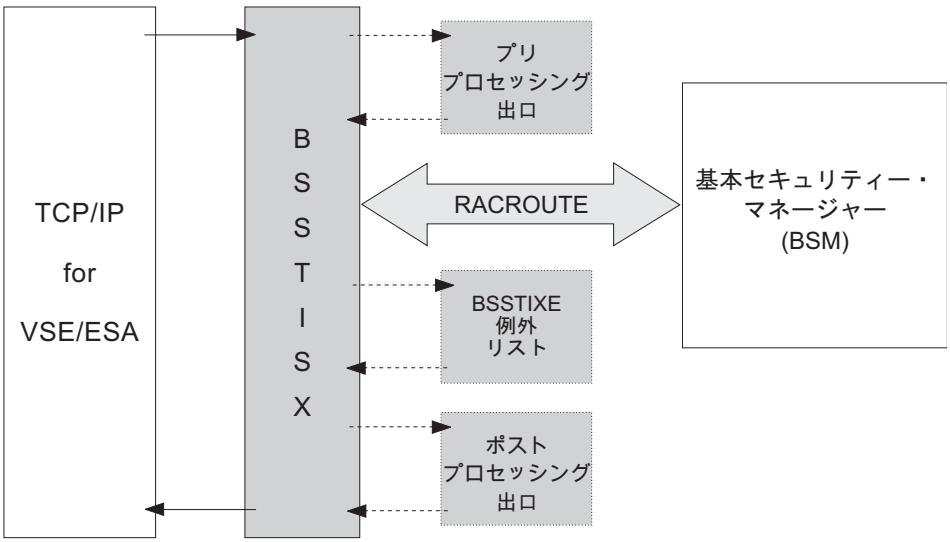
TCP/IP for z/VSE でのセキュリティー概念については、「*TCP/IP for VSE Installation Guide*」に説明があります。

VSE セキュリティーについての詳細は、「*IBM z/VSE* 計画」と「*IBM z/VSE* 管理」に記載されています。

セキュリティー機能の 1 つが、TCP/IP セキュリティー出口点です。これは、TCP/IP 提供のサンプル・コード SECEXIT を介して使用できます。しかし、TCP/IP 内部セキュリティー機能も、このサンプル出口コードも、VSE オペレーティング・システムのセキュリティー機能、つまり、基本セキュリティー・マネージャー (BSM) を活用しません。その結果として、お客様は、ユーザー ID および VSE リソースの定義と管理を 2 回行わなければなりません。1 回は TCP/IP において、もう 1 回は VSE オペレーティング・システムのセキュリティー・システムにおいてです。

このような状況を改善するために、TCP/IP 提供のサンプル・コード SECEXIT に代わる、フェーズ BSSTISX が導入されました。

以下の図に示すように、BSSTISX は、TCP/IP と BSM との間のリンクとして組み込まれています。



フェーズ BSSTISX は、BSM 機能を活用します。このフェーズは、RACROUTE 要求を発行することによって、ユーザー ID とユーザー認証を処理し、VSE ファイル、ライブラリー、およびメンバーのリソース・アクセス制御を処理します。また、POWER スプール・ファイルおよび SITE コマンドへのアクセス制御を制限することもできます。

POWER スプール・ファイルへのアクセスは、以下の条件が満たされる場合、管理者およびユーザーに許可されます。

- ユーザー ID が、要求されたスプール・ファイルの FROM ユーザー ID または TO ユーザー ID と一致するか、または、
- TO=ANY が指定された。

注: ANONYMOUS に割り当てられたユーザー ID には、これらのファイルへのアクセス権限はありません。

SITE コマンドを使用できるのは、管理者だけです。

TCP/IP 出口点を介して行うことができ、BSSTISX ではカバーされないような検査が他に多くあります。従って、BSSTISX には、プリプロセッシングとポストプロセッシングの出口インターフェースが備わっています。追加の検査を必要とするお客様は、BSSTISX に対して独自のプリプロセッシングとポストプロセッシングのルーチンを作成できます。

例外リスト BSSTIXE

出口 BSSTISX は、通常、この出口で評価できなかったすべてのアクセス要求を拒否します。しかし、特定の要求を拒否しないようにしなければならない場合があります。お客様は、そうした要求をこの例外リストに指定することができます。例外リストは、フェーズ BSSTIXE として、アセンブルし、リンクする必要があります (ライブラリー 59 の SKEXCLST を参照してください)。

IBM から配布されるフェーズおよび関連するソース・メンバー BSSTIXE.A は IJSYSRS.SYSLIB にあります。

要求は、SXBLOK フィールドである SXTYPE と SXFTYPE によって定義されます。SXBLOK は、TCP/IP と BSSTISX との間のインターフェースを記述するものです。SXBLOK のレイアウトは、TCP/IP for z/VSE と共に配布されます。

警告**:** 例外リストには、BSSTISX が評価できなかった要求のみを定義してください。例外リストに定義された要求に対しては、セキュリティー検査は実施されません。ご使用のシステムでのセキュリティー要件に影響を与えない要求のみを例外リストに追加するように注意してください。例外リストの代わりに、BSSTISX のプリプロセッシング出口とポストプロセッシング出口を使用して、ご使用のシステム特有のセキュリティー検査を追加することもできます。

セキュリティー出口の活動化

セキュリティー出口を活動化するには、次の TCP/IP コマンドを入力する必要があります。

DEFINE SECURITY,DRIVER=BSSTISX[,DATA='data']

DEFINE SECURITY コマンドは、セキュリティー出口 BSSTISX.PHASE を TCP/IP パーティションにロードします。

SET SECURITY =ON

SET SECURITY=ON コマンドは、セキュリティー処理を活動化し、初期設定のために BSSTISX に制御を渡します。BSSTISX は、追加パートをストレージにロードし、**data** に指定されたパラメーターに従って、制御ブロックを初期設定します。これ以降、TCP/IP は、出口ルーチン BSSTISX に検証のために情報を渡します。

SET SECURITY =ONX

セキュリティー出口 BSSTISX が FTPBATCH にも使用される場合は、ONX を指定します。

DEFINE SECURITY コマンドの **DATA=** パラメーターは、BSSTISX 用の初期設定パラメーターを含みます。構文は次のとおりです。

DATA='[anonym_uid][,[anonym_pwd][,[preproc][,[postproc]]]]'

anonym_uid

ここには、BSM に定義されるユーザー ID を指定できます。クライアントがユーザー ID ANONYMOUS でログオンするたびに、ここで指定したユーザー ID とそのアクセス権が使用されます。

anonym_pwd

このパラメーターで、ユーザー ANONYMOUS 用に BSM に定義されたユーザー ID のパスワードを指定できます。

preproc

自分で作成したプリプロセッシング出口を使用する場合、そのプリプロセッシング出口フェーズの名前をここに指定します。

postproc

自分で作成したポストプロセッシング出口を使用する場合、そのポストプロセッシング出口フェーズの名前をここに指定する必要があります。

セキュリティー出口の非活動化

セキュリティー出口を非活動化するには、次の TCP/IP コマンドを入力する必要があります。

SET SECURITY=OFF

SET SECURITY=OFF コマンドは、セキュリティー処理を停止し、クリーンアップおよび終了処理のために BSSTISX に制御を渡します。BSSTISX は、その制御ブロックをクリアし、追加パートのストレージを解放します。

DELETE SECURITY

DELETE SECURITY は、セキュリティー出口 BSSTISX.PHASE を解放します。

注: 新しいバージョンのセキュリティー出口を使用する場合、TCP/IP をシャットダウンし、再始動した後で、DEFINE SECURITY を入力する必要があります。

プリプロセッシング出口およびポストプロセッシング出口の使用

プリプロセッシング出口には、BSSTISX 初期設定の後に制御が渡され、それ以降の各要求の始めにも制御が渡されます。ポストプロセッシング出口に制御が渡されるのは、終了要求を除く各要求の最後です。両出口とも、TCP/IP が作成した SXBLOK から必要情報を取得します。

SXBLOK は、TCP/IP の出口点とセキュリティー出口との間のインターフェースを記述するものです。SXBLOK のマッピングは、TCP/IP for z/VSE と一緒に出荷されます。BSSTISX プリプロセッシング出口およびポストプロセッシング出口に使用する実際のレベルの、TCP/IP の SXBLOK を使用するように注意してください。

プリプロセッシング出口およびポストプロセッシング出口は、両方とも以下の要件を満たしていなければなりません。

- 再入可能
- AMODE(31)
- RMODE(24)

汎用レジスターの使用法を以下に示します。

プリプロセッシング出口のレジスター設定

入り口での設定:

R1 SXBLOK のアドレス

R13 標準保管域

R14 戻りアドレス

R15 プリプロセッシング出口フェーズのエントリー・ポイント

プリプロセッシング出口は、戻る前にレジスターをリストアしなければなりません。レジスター 15 が次のように結果を示します。

戻るとき:

R15 = 0
BSSTISX は通常の処理を続行するべきである

R15 = 'E0'x
BSSTISX は、すべての検査をスキップし、R15=0 (違反なし) で終了するべきである

R15 = 4
BSSTISX は、すべての検査をスキップし、R15=4 (セキュリティー違反) で終了するべきである

ポストプロセッシング出口のレジスター設定

入り口での設定**:**

R0 BSSTISX の現在の戻りコード値

R1 SXBLOK のアドレス

R2 BSSTISX からの理由コード

R13 標準保管域

R14 戻りアドレス

R15 ポストプロセッシング出口フェーズのエントリー・ポイント

ポストプロセッシング出口は、戻る前にレジスターをリストアしなければなりません。レジスター 15 が次のように結果を示します。

戻るとき**:**

R15 = 0
BSSTISX は R15=0 (違反なし) で終了するべきである

R15 = n
BSSTISX は R15=n で終了するべきである。n=4 はセキュリティー違反を表します。

R15 = 4
BSSTISX は、すべての検査をスキップし、R15=4 (セキュリティー違反) で終了するべきである

ポストプロセッシング出口に渡される BSSTISX からの理由コードは、次のとおりです。

X'00' 特定の理由コードなし

X'10' アクセスは許可された - ユーザーは管理者である

X'11' アクセスは例外リスト項目によって許可された

X'12' アクセスは RACROUTE AUTH 要求によって許可された

X'13' アクセスは許可された - ICCF オプションが指定された

X'14' アクセスはプリプロセッシング出口によって許可された

X'15' アクセスは許可された - OPEN によって検査される

X'16' アクセスは BSSTISX によって常に許可される

X'17' アクセスは POWER によって許可された。FROM/TO または ANY ユーザーである。

X'18' アクセスは、マスター・コンソールをオープンする権限によって許可された

X'19' アクセスは許可された。$NULL ファイル要求である。

X'20' アクセスは拒否された - ユーザーは管理者ではない

X'21' アクセスは拒否された - サポートされない要求

X'22' アクセスは RACROUTE AUTH 要求によって拒否された

X'23' アクセスはオプション・コードが 4 であるため拒否された

X'24' アクセスはオプション・コードが 8 であるため拒否された

X'25' アクセスは内部エラーのため拒否された

X'26' アクセスはプリプロセッシング出口によって拒否された

X'27' アクセスは拒否された。POWER への読み取り要求ではない。

X'28' アクセスは POWER によって拒否された。FROM/TO または ANY ユーザーでない。

パフォーマンス改善のヒント

TCP/IP の使用法によっては、BSSTISX は同じユーザー ID で多数のユーザー検査を実行しなければならないことがあります。このような状態の場合、以下を介して BSM キャッシュを活動状態にすることが役立ちます。

```
MSG xx,DATA=DBSTARTCACHE
```

ここで、*xx* は、セキュリティー・サーバー・パーティションのパーティション ID を表します (デフォルトは FB)。

外部セキュリティー・マネージャー

TCP/IP セキュリティー出口 BSSTISX は、外部セキュリティー・マネージャー (External Security Manager (ESM)) と一緒に使用することもできます。ただし、それらの ESM が、BSSTISX によって発行される RACROUTE 要求をサポートしている場合に限ります。例えば、CA Inc. が配布する CA-Top Secret は、そういった RACROUTE 呼び出しをサポートします。

第 5 章 TCP/IP for z/VSE の Infoprint Manager サポート

InfoPrint Manager (IPM) サポートによって、EBCDIC モードの印刷ファイルを、IPM が稼働している AIX® または Windows ワークステーションに転送することができます。EBCDIC 転送であるため、このサポートでは印刷対象の文書内で制御文字が維持されます。このサポートの前提条件については、 35 ページの『ソフトウェア前提条件』に説明があります。

下の表は、AIX、Windows NT、Windows 2000、または Windows XP 上の InfoPrint Manager (IPM) に提供されているサポートの概要を示しています。

サポート	**LPR** スクリプト	**$$ LST** 指定*
EBCDIC	SET INFOPRINT=YES/NO (または =ON/OFF)	n/a
Pagedef	SET PAGEDEF=pdef [1]	PAGEDEF=pdef [1, 2]
Formdef	SET FORMDEF=fdef [3]	FORMDEF=fdef [3, 4]
Forms	SET FNO=fno [5]	FNO=fno [5]

1 `pdef=` 英数字の最大文字数は 6 です

2 POWER 生成に次の定義が必要です。

```
DEFINE L,PAGEDEF,1F,1,6,C
```

3 `fdef=` 英数字の最大文字数は 6 です

4 POWER 生成に次の定義が必要です。

```
DEFINE L,FORMDEF,1D,1,6,C
```

5 `fno=` 英数字の最大文字数は 4 です

* AUTOLPR でのみ使用可能

[1] と [3] については、実際の値は z/VSE において POWER コマンド `D AUSTMT` でチェックできます。

注**:**

1. EBCDIC サポートは、LPR/LPD を使用することでのみ有効になります。
2. $$ LST と SET に値が定義された場合、SET コマンドからの値が使用されます。
3. サポートされる英数字は、A-Z、0-9、#、@、$ です。

このサポートの使用例を以下に示します。

```
* $$ JOB JNM=TRLTSTPR,CLASS=0,PRI=5,USER=TEST
* $$ LST CLASS=L,DEST=(*,ANY)
// JOB TRLTSTLPR    *** LPR to AIX queue ***
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD2.TCPIPC,PRD2.SCEEBASE)
// EXEC CLIENT,PARM='APPL=LPR,ID=00'
SET HOST=9.66.110.67
SET PRINTER=ipheft
SET PAGEDEF=b111
SET FORMDEF=a222
SET INFOPRINT=YES
SET CC=YES
SET CRLF=UNIX
SET NOEJECT=ON
SET DISP=KEEP
PRINT POWER.LST.L.TRLUH003
QUIT
/*
/&
* $$ EOJ
```

図 *11. TCP/IP for z/VSE* 上での *LPR* ジョブ

IPM サポートのパラメーター設定

SET パラメーターの説明

SET HOST=9.66.110.67

この設定によって、IP アドレス 9.66.110.67 で Infoprint Manager を搭載するシステム (AIX または Windows) がアドレッシングされます。特定のプリンターの IP アドレスではなく、システムがアドレッシングされることに注意してください。

SET PRINTER=ipheft

この設定によって、ipheft という名前の論理宛先 (LD) がアドレッシングされます。InfoPrint Manager の論理宛先は、一般的にプリンター待ち行列に例えることができます。AIX では、この ipheft という名前の実際の AIX プリンター待ち行列は存在していないはずであり、もし存在していれば、この待ち行列が IPM の LD の代わりにアドレッシングされることに注意してください。

LPR では、AIX の LPD がアドレッシングされます。LPD は、その特定の名前 (ipheft) のプリンター待ち行列を認識していないので、LPD は印刷ジョブを自動的に IPM にルーティングします。そうすると、IPM はその名前 (ipheft) を既知の LD として認知して LPD に戻し、印刷ジョブをこの LD に割り当てます。

SET PAGEDEF=b111 および SET FORMDEF=a222

これらの設定によって、AFP 印刷フォーマット設定に必要な PAGEDEF 名と FORMDEF 名を指定できます。これらの定義は、前に P1 または F1 を付けて自動的に拡張され、アドレッシングされた LD 内の PAGEDEF および FORMDEF のデフォルト指定を上書きします。これらの PAGEDEF および FORMDEF リソースは、IPM サイトで使用可能でなければなりません。図 11 の例では、P1b111 という PAGEDEF 定義は IPM サイトで使用可能でなければなりません。

SET INFOPRINT=YES

この設定によって、印刷データおよびそれに関連する制御文字の伝送が

EBCDIC で行われます。これによって、マシン・コードまたは ASA コード、AFP 構造化フィールドの送信が可能になります。AFP コマンド (x'5A' データ・レコード) を組み込むことができ、それによって、例えば、IMM コマンドで COPYGROUP を直接呼び出したり、BDT/EDT コマンドを使用して個々のステープル止めを制御できます。

SET CC=YES
この設定は、印刷制御文字または AFP 制御文字 (x'5A') を送信するために必要です。

SET CRLF=UNIX、SET NOEJECT=ON、SET DISP=KEEP
これらのパラメーターは、データ転送が正しく行われ、送信データ・セットがファイル属性指定 KEEP に入れられるようにするために必要です。

IPM での SET FNO= パラメーターの使用

フォーム・パラメーター SET FNO=fno を使用するには、以下を定義する必要があります。

1. IPM のファイル /etc/environment に、次の 1 行を組み込む: PD_FORMS=true
2. IPM が使用する実際の宛先を準備する。これは、次のセクションで説明するように、実際の宛先に関して 2 つのパラメーター変更を行うことによって実行します。

Infoprint Manager のカスタマイズ

AIX または Windows システム上の Infoprint Manager には、論理宛先 (LD) が定義されている必要があります。 32 ページの図 11 に示した例では、この LD は ipheft という名前です。

この LD の文書デフォルトは、次のように定義される必要があります。

Document Other: Format = line-data
この定義は、LPR または LPD 通信を使用する行データ転送のために必要です。これは、AFP 構造化フィールド・レコードが埋め込まれた行データ、つまり、混合モード AFP 印刷アプリケーションにも有効です。

Document Processing: Transform Options = INDEXOBJ=BDTLY
この設定は、例えば、IP2000、IP60、IP70 などの IBM プリンターを使用する際にステープル止めが必要な場合にのみ使用するべきです。そうしないと、ACIF を使用した索引付けができなくなることに注意してください。

Document Line Data: Location of page definitions = /usr/lpp/psf/user/ppfalib
このディレクトリーは、例えば、定義された PAGEDEF リソースがある場所を指定します。

Type of carriage control characters = machine
この定義は、マシン・コードまたは ASA コードのどちらの印刷制御コードが使用されるのかを指示します。

Convert to EBCDIC = No
この定義は、データが EBCDIC で転送されることを指定します。

Document AFP Resources: Location of form definitions = /usr/lpp/psf/user/ppfalib
このディレクトリーは、例えば、定義された FORMDEF リソースがある場所を指定します。

実宛先のプロパティーの変更

実際の宛先のプロパティーを変更するため、以下を定義します。

```
Load Balancing: Disable on Job mismatch = No
```

```
Load balancing: Job-Batches-Ready ADD = fno-value
```

(*fno-value* の最大文字数は 4 文字です)

注: この実際の宛先について、いくつかの fno-value を追加することができます。

fno-value=Job-Batches-Ready-value が一致するジョブを IPM が受け取ると、その印刷ジョブは即時に印刷を開始します。IPM のオペレーターは、印刷ジョブに job-batch-value=fno-value が割り当てられていることを確認します。一致する fno-value=Job-Batches-Ready-value がないジョブを IPM が受け取った場合、その印刷ジョブは保留状態になり、*'resources not ready'* を示すメッセージが表示されます。このような場合、オペレーターは、プリンターに要求された用紙をセットし、実際の宛先の Job-Batches-Ready-value を要求されたものに変更し、実際の宛先を作動可能に設定する必要があります。保留中の印刷ジョブは自動的に開始します。

オペレーターは、実際の宛先を右クリックして Job-Batches-Ready を選択するというショートカットを使用することもできます。必要に応じて、値の ADD または REMOVE を選択します。VSE システムから受け取った fno-value は、印刷ジョブの JOB-BATCH 値に移され、それに従って表示されます。印刷ジョブの JOB-BATCH 値は、最後に、実際の宛先の Job-Batches-Ready 値と比較され、それに応じて IPM が動作します。さらに、パラメーター fno-value が -opassthru=forms 値に渡され、ジョブ・セパレーター・シートに 'FORMS: fno-value' として印刷されます。

バックグラウンド技術情報

SET INFOPRINT=YES が使用された場合、TCP/IP for z/VSE の LPR は、パラメーター -ofileformat=record も転送します。この設定によって、すべてのレコードの先頭に 2 バイト長のフィールドが付けられます。このフィールドは IPM によって自動的に検出され、印刷の前に除去されます。

転送されるファイルの LPD/LPR 制御ファイル内に、以下の設定が見つかる場合があります。例えば、TCP/IP トレースの検査中などに見つかります。

サポート	**LPR** スクリプト	変換後の **-o** オプション
EBCDIC	SET INFOPRINT=YES/NO (または =ON/OFF)	-ofileformat=record および -odatatype=line
Pagedef	SET PAGEDEF=pdef	-opagedef=P1pdef
Formdef	SET FORMDEF=fdef	-oformdef=F1fdef

サポート	LPR スクリプト	変換後の -o オプション
Forms	SET FNO=fno	-opassthru=forms=fno および -oforms=fno

ソフトウェア前提条件

InfoPrint Manager サポートは、最小限、以下のソフトウェア・レベルを必要とします。

- IPM for AIX 3.2、APAR IY17446 / PTF U475406、または

  IPM for Windows NT または Windows 2000 1.1、CSD レベル 1.1.0.10

第 6 章 TCP/IP for z/VSE がサポートする z/VSE 関連ハードウェア機能

TCP/IP for z/VSE でサポートされるハードウェア関連機能は、以下のとおりです。

- ハードウェア暗号化
- HiperSockets
- OSA Express2 および OSA Express

ハードウェア暗号化サポート

z/VSE ハードウェア暗号化支援サポート (ハードウェア暗号化サポートと呼ばれる) には、Crypto Express またはそれと同等の Crypto カードが必要です。これらのカードは、IBM Z 環境で使用できます。これは、暗号化支援サポートを提供し、SSL/TLS (Secure Sockets Layer / Transport Layer Security) を使用する TCP/IP ネットワークにおけるスループットの増大に寄与します。

z/VSE が z/VM 下で稼働する場合、z/VM 4.2 以上が必要です。

詳しくは、「*IBM z/VSE* 計画」を参照してください。

HiperSockets

z/VSE は、HiperSockets を使用した、論理区画 (LPAR) および仮想マシン間での高速 TCP/IP 通信をサポートします。HiperSockets のサポートは、IBM Z 環境で利用できます。

詳しくは、「*IBM z/VSE* 計画」を参照してください。

OSA Express サポート

OSA Express サポートは、IBM Z 環境における統合ハードウェア機能 (OSA Express3、OSA Express2、および OSA Express アダプター) として提供されます。このサポートは、z/VSE アプリケーションと、接続されたネットワーク内の他のプラットフォームとの間の直接接続を提供します。これは、Queued Direct Input/Output (QDIO) アーキテクチャーがベースであり、効率的なデータ転送を可能にすることによって TCP/IP データ・パケット転送の高速化を実現します。

詳しくは、「*IBM z/VSE* 計画」を参照してください。

OSAX デバイス・ドライバーの拡張構成については、 619 ページの『付録 C. 拡張 OSAX デバイス・ドライバーの構成』を参照してください。

第 7 章 パフォーマンス考慮事項

パフォーマンス関連パラメーターの変更

パフォーマンスに関するセットアップまたは操作パラメーターの設定で最適の選択を行うことは非常に重要です。以下の 2 種類のデフォルトがあります。

製品デフォルト
: 製品デフォルトは、値が明示的に割り当てられていない場合に適用されます。

出荷時デフォルト
: 出荷時デフォルトは、お客様が TCP/IP for z/VSE の出荷時の始動ジョブのスタートアップ値を、変更または上書きしていない場合に適用されます。パラメーターに対する出荷時の特定のスタートアップ値は、通常、スタートアップ点として使用するのに適した値を表しています。ただし、具体的な負荷または構成によっては、変更が妥当である場合もあります。

両者の値が同じでないことはよくあります。ワークロードに影響しないパラメーターを変更しても変化を確認できないので、変更する前によく注意してください。

多くのパラメーターが TCP/IP for z/VSE パフォーマンスに影響を与えますが、具体的な稼働環境に合わせてそれらの値を調整できます。

一般的に、パラメーターによるチューニングは、次のようにグループ分けできます。

- オペレーティング・システムのチューニング
- TCP/IP のチューニング
- 通信のチューニング (メインフレーム終端およびワークステーション終端)
- TCP/IP アプリケーションのチューニング

z/VSE のほとんどのお客様にとってオペレーティング・システムのチューニングは新しいことではないので、ここでは詳しくは説明しません。

TCP/IP のチューニングの効果についてよく理解するには、基本的な TCP/IP 概念を把握することが役立ちます。これらの概念には、以下のものがあります。

- フレーム、データグラム、およびセグメント
- フラグメント化および再組み立て
- ウィンドウ・サイズと肯定応答による、送受信バッファーの管理

通信のチューニングは、TCP/IP for z/VSE のチューニングと密接に関連しています。通信のチューニングは、ネットワーク構成 (リンク等を含む) に加えて、相手側 (これも TCP/IP です) のパラメーター選択も参照します。

他のタイプのチューニングでも同じですが、変更は一度に 1 つだけ行うようにしてください。ご使用の環境において、あるパラメーターを変更しても、別の値がパフ

ォーマンスを支配しているために、まったく改善が見られないことがあります。その値を変更すると、同じ変更内容でもパフォーマンスがかなり改善することがあります。

一般的なパフォーマンス問題

次のタイプのパフォーマンス・データがあります。

アクティビティーのリソース使用量
: 特定の TCP/IP アクティビティーを実行するために必要な CPU 時間、入出力量。例えば、CICS トランザクションのための Telnet の使用や、1 M のデータの転送などです。

達成可能なスループット/パフォーマンス値
: TN3270 で並行してサポート可能な端末の数、または、ある特定の環境において 1 つの並行 FTP アクティビティーに対して達成可能なデータ転送速度。

TCP/IP for z/VSE の主要なパフォーマンス依存関係

TCP/IP アプリケーションを使用しているときのパフォーマンスは、関係するハードウェアおよびソフトウェア・プロダクトに大きく依存します。以下の表は、主要なパラメーターのリストです。このリストはパフォーマンス/チューニングの影響を、全体的に分類することを目的としています。ここに示された構成要素によって、全体的なパフォーマンスが決まります。

表 *1.* パフォーマンスに関する主要パラメーター

パラメーター (タイプ)	ホスト **CPU** 時間	ホスト・ストレージ	転送時間	**DASD** 時間
ホスト CPU 速度	X	-	-	-
IBM Z システム制御プログラムおよびセットアップ	X	X	-	x
使用 MTU/MSS	X	x	X	-
ウィンドウ・サイズ	-	x	X	-
転送バッファー数	-	X	x	-
通信アダプターのタイプ	-	-	X	-
ネットワーク/回線速度	-	-	X	-
ネットワーク信頼性	X	x	X	-
アプリケーションの入/出バイト数	X	X	X	X (アプリケーションに依存)
TCP/IP インプリメンテーション	X	X	X	X
TCP/IP アプリケーション	X	X	X	X
他の TCP/IP パラメーター	X	X	X	X
DASD I/O サブシステム	-	-	-	X
DASD I/O ブロッキング	x	-	-	X

表 *1.* パフォーマンスに関する主要パラメーター *(*続き*)*

パラメーター **(**タイプ**)**	ホスト **CPU** 時間	ホスト・ストレージ	転送時間	**DASD** 時間
注: **X** 大きな影響があることを示します。 **x** 比較的小さな、あるいは 2 次的な影響があることを示します。 **-** 影響がまったくないか、あるいは無視できることを示します。 転送時間には、転送を待つ時間も含まれます。 **DASD** 時間は、**DASD** が関係する場合 **(**例えば、**FTP)** のみ該当します。				

全体的なキャパシティーも関心の対象であり、これは複数並行セッション (例えば、Telnet3270) の場合には特に重要です。

以下の表は、主要なパラメーターのリストであり、TCP/IP for z/VSE におけるパフォーマンス関連の設定を示しています。また、特定のパラメーターがどの TCP/IP アクティビティーに影響するのかも示しています。

表 *2. TCP/IP* パフォーマンス関連パラメーター

TCP/IP アクティビティーの有効範囲					
TCP/IP パラメーター**/**設定	任意	アウトバウンド	**TCP** インバウンド	**TN3270** アウト **+** イン	**FTP** アウト **+** イン
DEFINE ADAPTER | LINK MTU TELNETD POOL		X		X	
SET ALL_BOUND REDISPATCH ARP_TIME REUSE_SIZE FULL_SCAN GATEWAY CHECKSUM	X X X x X X x				
Set MAX_SEGMENT WINDOW_DEPTH CLOSE_DEPTH WINDOW_RESTART			X1 X1 X1 X1		
SET RETRANSMIT FIXED_RETRANS WINDOW ADDITIONAL_WINDOW		X1 x1 X1 x1			
SET TELNETD_BUFFERS TRANSFER_BUFFERS MAX_BUFFERS				X2	X X
x 比較的小さな、あるいは 2 次的な影響があることを示します。 **X1** TCP 負荷についてのみ (FTP を含むが、NFS は含まない) **X2** POOL=YES TELNET デーモン/セッションについてのみ					

より具体的な TCP/IP for z/VSE パフォーマンス情報およびパフォーマンス結果については、z/VSE インターネット・ホーム・ページ ***http://www.ibm.com/systems/z/os/zvse/*** から参照できます。

個々のコマンドの操作およびデフォルト値の説明については、「*TCP/IP for VSE Command Reference*」を参照してください。

第 2 部 IPv6/VSE の使用

第 8 章 IPv6/VSE 概要

この資料の 43 ページの『第 2 部 IPv6/VSE の使用』では、IPv6/VSE (5686-BS1) プログラムを概説しています。IPv6/VSE は、z/VSE 向けの IPv6 ソリューションを提供する、TCP/IP (Transmission Control Protocol/Internet Protocol) のネイティブ実装です。

IPv6/VSE は、IPv6 TCP/IP スタック、IPv6 アプリケーション・プログラミング・インターフェース (API)、および IPv6 対応アプリケーションを提供します。
| IPv6/VSE V1R3 は、IPv4 および IPv6 プロトコルをサポートします。

IPv6/VSE の IPv6 TCP/IP スタックは、1 つの z/VSE システム内で、IPv4 TCP/IP スタックと同時に実行できます。

IBM は、このプログラムとともに使用できる拡張機能を提供しています。こうした EZASMI マクロ・インターフェースや EZASOKET コール・インターフェースなどの拡張機能は、詳細に説明されています。こうした外部インターフェース内の拡張機能について、実際に IPv6/VSE プログラムが提供するよりも多くの機能が説明されている場合があることに注意してください。そのような例外については、IPv6/VSE の資料を確認してください。

OSAX デバイス・ドライバーの拡張構成については、 619 ページの『付録 C. 拡張 OSAX デバイス・ドライバーの構成』を参照してください。

IPv6/VSE プログラムをご使用になる前に、以下の項目を熟読してください。

IPv6 TCP/IP スタック

IPv6/VSE の IPv6 TCP/IP スタックは、固有のスタック ID を使用して、別のパーティションで実行します。これにより、IPv4 および IPv6 の両方のスタックは、1 つの z/VSE システム内で同時に実行できます。別個の IPv4 および IPv6 スタックを 1 つの z/VSE システム内で同時に実行すると、パフォーマンスと信頼性の両面で向上を図ることができます。既存の IPv4 アプリケーションは、未変更のまま IPv4 TCP/IP スタックを使用して引き続き実行できるので、お客様の既存の投資は保護され、活用されます。新しい IPv6 対応アプリケーションは、IPv6/VSE の IPv6 スタックを使用して、段階的に導入できます。

デュアル・スタック・サポート

デュアル・スタック・サポートを使用すると、アプリケーションは IPv4 および IPv6 の両方のネットワークに同時に接続できます。デュアル・スタック・サポートの実装により、単一の IPv6 対応 CICS トランザクションまたはバッチ・アプリケーションは、IPv4 または IPv6 ネットワークのいずれかによりパートナーと通信できます。IPv6 対応アプリケーションの導入に使用できる、拡張ソケット API が提供されています。

IPv6 対応ユーティリティー・アプリケーション

IPv6/VSE は、IPv6/VSE スタック・パーティション外で実行するユーティリティー・アプリケーションを提供しています。IPv6/VSE スタック・パーティション外のこれらのアプリケーションにより、安定度とパフォーマンスを向上させることができます。

FTP サーバー
: IPv6/VSE FTP サーバーは、リモート・ホスト FTP クライアントにより、POWER キュー、VSAM カタログ、SAM ファイル、および z/VSE ライブラリーなどの、z/VSE リソースへのアクセスをサポートします。

バッチ **FTP** クライアント
: IPv6/VSE バッチ FTP クライアントは、リモート・ホスト FTP サーバーへのアクセスを提供する、z/VSE バッチ・ジョブとして実行します。データは、それらのリモート FTP サーバーと交換できます。

TN3270E サーバー
: IPv6/VSE TN3270E サーバーは、TN3270/TN3270E 端末セッションおよび TN3270E プリンター・セッションをサポートします。加えて、DIRECT、LPR、および FTP プリンター・セッションがサポートされています。

NTP サーバー
: IPv6/VSE NTP サーバーは、Network Time Protocol サーバーであり、リモート・ホストは z/VSE の時刻 (TOD) クロックを照会して、そのクロックと z/VSE クロックを同期させることができます。

NTP クライアント
: IPv6/VSE NTP クライアントにより、z/VSE はその TOD クロックを外部ソースに設定できます。

システム・ロガー・クライアント
: このアプリケーションは、選択した z/VSE コンソール・メッセージを、リモート Linux syslog-ng デーモンにログ記録するために使用します。メッセージを syslog-ng デーモンに送信すると、Linux 自動化プロセスを使用して、イベントを起動できます。

バッチ **E** メール・クライアント
: IPv6/VSE バッチ E メール・クライアントは、E メールを SMTP サーバーに送信するために使用します。これに応じて、SMTP サーバーは E メールを宛先ユーザーに送信します。宛先の数は制限されておらず、発信 E メール・メッセージにはファイルを添付できます。

バッチ **LPR**
: IPv6/VSE バッチ LPR アプリケーションは、POWER キューからデータを取り出し、それをリモート・ホスト LPD に転送します。LPD は、プリンターに常駐させることも、リモート・ホスト上でサーバーとして実行することもできます。

バッチ・リモート実行クライアント
: IPv6/VSE リモート EXEC クライアントを使用すると、z/VSE パーティシ

ョンで実行するジョブは、スクリプトを起動してリモート・ホスト上で実行できます。スクリプトからのすべての出力はクライアントに返され、完了情報についてスキャンされます。

バッチ **PING**
IPv6/VSE バッチ PING アプリケーションは、リモート・ホストを ping するために使用されます。

GZIP データ圧縮
IPv6/VSE は、簡易な gzip データ圧縮アプリケーションを提供しています。データは SAM ファイルまたはライブラリー・メンバーから読み取ったり、圧縮したり、それらに書き込んだりできます。圧縮データは、リモート・ホストに転送して処理することができます。この逆の処理も実行できます。

REXX 自動化
IPv6/VSE は、自動化のために z/VSE REXX EXEC を使用します。データの自動 FTP は、提供されているサンプルの REXX EXEC で処理されます。データの自動 LPR または自動 E メールも同じ方法で処理されます。IPv6/VSE アプリケーションを REXX EXEC を使用して呼び出すことにより、コマンドや、例えばファイル名や日付などのパラメーターを動的に作成できます。

BSTTPRXY/BSTTATLS
IPv6/VSE は、z/VSE のサーバーおよびクライアント・アプリケーション用の SSL/TLS を提供する 2 つのサーバーを提供します。この 2 つのサーバーは、アプリケーションに透過的に、SSL/TLS を提供します。また、この 2 つのアプリケーションは、ASM SOCKET マクロ、EZASMI、EZASOKET、および LE/C API を使用するアプリケーションを含め、サポートされる任意の API で作成されたバッチおよび CICS アプリケーションをサポートします。

- SSL プロキシー・サーバー (BSTTPRXY) は単純なプロキシー・サーバーです。これは、単一の PROXY コマンドのみを許可します。複数の接続をプロキシー処理するには、複数の BSTTPRXY 区画を実行する必要があります。BSTTPRXY は、IPv4 から IPv6 または IPv6 から IPv4 への変換を実行します。
- AT-TLS サーバー (BSTTATLS) の Automatic Transport Layer Security は、z/OS の AT-TLS (Application Transparent - Transport Layer Security) 機能に似た機能です。BSTTATLS では、多くの AT-TLS 定義を使用でき、これを使用して着信および発信接続を監視し、必要に応じてソケットを代行受信して平文から SSL またはその逆に変換できます。ただし、BSTTATL では、IPv4 から IPv6 または IPv6 から IPv4 への変換は実行されません。

IPv6/VSE ユーティリティー・アプリケーションの FTP サーバー、FTP クライアント、LPR、バッチ E メール・クライアント、および GZIP は、2 バイト文字セット (DBCS) をサポートします。

IPv6/VSE (5686-BS1) プログラムの資料

IPv6/VSE 製品 (IBM 製品番号: 5686-BS1) の製品説明は、z/VSE DVD コレクション (SK88-8070) およびインターネット (http://www.ibm.com/systems/z/os/zvse/ 資料) から、PDF 形式でのみ入手できます。

IPv6/VSE の製品資料には、IPv6/VSE 製品のプロバイダーである Barnard Software Inc. によるオリジナルの製品説明を記載した 8 種類の資料に加えて、IBM が提供する IPv6/VSE 製品の概要を示した資料があります (本書)。

資料は次のとおりです。

- *z/VSE TCP/IP サポート* (この資料)
- *IPv6/VSE IPv6 Installation Guide*
- *IPv6/VSE IPv4 Installation Guide*
- *IPv6/VSE IPv6 User's Guide*
- *IPv6/VSE IPv4 User's Guide*
- *IPv6/VSE Programming Guide*
- *IPv6/VSE Migration Guide*
- *IPv6/VSEMessages and Codes*
- *IPv6/VSE SSL Installation, Programming and User's Guide*

Adobe Reader を使用して、これらの資料を表示および印刷できます。Adobe Reader がまだインストールされていない場合、または Adobe Reader のインストールおよび使用方法についての情報が必要な場合は、Adobe Web サイト (http://www.adobe.com) を参照してください。

IPv6/VSE のインストール要件

プロセッサー要件

IPv6/VSE は、z/VSE 6.1 以降でサポートされるすべてのハードウェア構成で実行します。

予防サービス計画

IPv6/VSE をインストールする前に、入手可能な追加サービスであるかどうかについて、IBM サポート・センターに問い合わせるかまたは VSE ホーム・ページを参照してください。

ユーザー・アクセス・キー

IPv6/VSE (z/VSE のネイティブ IPv6 ソリューション) は、オプションで有料の IBM プログラム (5686-BSI) です。IBM は、このプログラムのライセンス交付を Barnard Software, Inc. 社から受けました。

IPv6/VSE は、単体の製品としても入手可能です。z/VSE 6.1 から、IPv6/VSE は z/VSE ベースで、ライブラリー PRD2.TCPIPB にプリインストールされています。

IPv6/VSE は、アクティベーション後の 30 日間は、キーなしで使用できます。30 日の試用期間の後には、IPv6/VSE はマシンの CPUID に応じた固有のユーザー・アクセス・キーが必要になります。

固有のユーザー・アクセス・キーを要求するには、IBM Copenhagen Key Center に、E メール (speckeys@dk.ibm.com) または電話 (+45 48 10 15 30) (通話料有料) にてお問い合わせください。Key Center から E メールでライセンス・キーを受け取った場合は、次の例に示すとおりに BSTTPARM.A フェーズを作成して以下の行を挿入し、個人データを入力して完成させる必要があります。

```
COMPANY name
CPUID xxxxxx MODEL xxxx
LICENSE TCP/IP-TOOLS ABCDEFGHL6Z date vcode
*
TCP/IP-TOOLS ENABLE
```

詳細なインストール手順については、「IPv6/VSE Installation Guide」を参照してください。

IPv6/VSE によるセキュリティー・マネージャーの活用

IBM は、BSSTISX と呼ばれるセキュリティー出口ルーチンを提供しています。IBM セキュリティー出口を呼び出すように BSI FTP サーバーのセキュリティー出口ルーチン BSTTFTS1.PHASE をセットアップして、ユーザー ID とパスワードを検証することができます。詳しくは、「IPv6/VSE User's Guide」を参照してください。

IPv6/VSE がサポートする z/VSE 関連ハードウェア機能 - プロセッサー要件

IPv6/VSE BSTT6NET TCP/IP スタックは、「*IBM z/VSE* 計画」の『ハードウェア・サポート』に記載されたプロセッサーを必要とします。

ネットワーク・インターフェースの要件

IPv6/VSE は CTCA、6in4 トンネル、OSA Express、および HiperSockets ネットワーク・インターフェースをサポートします。QDIO モードまたは HiperSockets ネットワーク・インターフェースで実行される OSA Express アダプターのみがサポートされます。

IPv6/VSE のファイアウォール

IPv6/VSE には、基本的なファイアウォール・セキュリティー機能があります。

このファイアウォールは、IPv4 および IPv6 のイーサネット IP パケットについて基本タイプの情報を調べます。ソース IP アドレス、パケット・プロトコル、TCP または UDP のポート番号、および ICMP メッセージ・タイプとコードを検査できます。この処理は、許可することも拒否することもできます。ファイアウォールは、ファイアウォール規則テーブルを含む、ライブラリー・メンバー BSTTFWRT.T により制御されます。スタックは、`LIBDEF SOURCE,SEARCH` チェーンを介してこのメンバーにアクセスします。

デフォルトでは、ファイアウォールは有効になっています。ただし、デフォルトのファイアウォール・ルールにより、すべてのパケットはスタックで処理できます。

パケットは、スタックによる処理を拒否されると除去されます。これは、「ステルス」モードまたはファイアウォール・ステルス操作と呼ばれています。パケットへの応答は生成されません。ファイアウォールを無効にするには、ライブラリー・メンバー BSTTFWRT.T を削除します。 BSTTFWRT.T メンバーを削除すると、始動時にスタックの SYSLST ログにエラー・メッセージが書き込まれます。 IPv6 を指定したファイアウォールのコマンドは、BSTTINET IPv4 スタックでは無視されます。 IPv4 を指定したコマンドは、BSTT6NET IPv6 スタックでは無視されます。TCP、UDP、および ICMP のルールは、両方のスタックに共通です。

BSTT6NET IPv6 スタックのファイアウォールは、常時さまざまな ICMPv6 トラフィックを許可します。このトラフィックは、IPv6 スタックが機能するために必要です。このトラフィックには、リンク・ローカル IPv6 アドレスへのパケットと、隣接者探索およびルーター/ゲートウェイ処理に必要とされるさまざまな ICMPv6 トラフィックが含まれます。 IP PROT、IP6 PROT、TCP、UDP、および ICMP の場合、ゼロの値は「すべての値と一致」を示します。標準的なファイアウォール規則は、「IPv6/VSE IPv4 Installation Guide」に示されている一連のコマンドに従う必要があります。

PROT の値は、6 (TCP)、17 (UDP)、ICMP (1)、および ICMPv6 (58) です。現在のところ、インバウンド (IN) 規則のみが処理されます。アウトバウンド (OUT) 規則の構文は、定義されていますが、現時点では実装されていません。

ファイアウォール構成、コマンド、および基本的なファイアウォール規則について詳しくは、「IPv6/VSE IPv4 Installation Guide」を参照してください。

第 3 部 プログラミング・インターフェース

第 9 章 ソケット・プログラミングの概要

このセクションでは、z/VSE のサポート対象の TCP/IP スタックで提供されるソケット・プログラミングの概要を説明します。

VSE ホストに接続して、異なるアクセス方式を使用するシステムとデータを交換できます。ご使用の TCP/IP スタックの製品情報を調べて、サポートされるアクセス方式を確認してください。

Telnet
: Telnet は、ローカル z/VSE 上で実行している VTAM アプリケーションに接続するために、リモート・ホストから使用できます。ローカル z/VSE ホスト上では、Telnet デーモンを実行する他のリモート・システム、例えば UNIX ワークステーションに接続するために使用できます。

ファイル転送プロトコル **(FTP)**
: FTP は、リモート・ホスト・システムとデータ・ファイルのやり取りを行うために使用されます。

Web サーバー
: Web ブラウザーを使用して Web サーバーにアクセスし、HTML (ハイパーテキスト・マークアップ言語) ページで定義されたデータを取得できます。

  - 静的ページ・コンテンツ: HTML のみ
  - 動的ページ・コンテンツ: JavaScript、Java™ アプレット、または呼び出し CGI (コモン・ゲートウェイ・インターフェース) プログラムを含む HTML

クライアント/サーバー・アプリケーション
: 分散アプリケーションは、企業イントラネットまたはインターネットを介して通信します。アプリケーションは、TCP/IP ソケット・プログラミング・インターフェースを活用して、対等通信を確立します。

このセクションでは、TCP/IP ソケットをベースにしたクライアント/サーバー・アプリケーションの要件にフォーカスを合わせます。使用するプログラミング・インターフェースとその使用方法を決定する前に、どのような観点の検討が必要なのかを示すことが本章の目的です。

TCP/IP ソケット接続とは何か?

ソケット・プログラミング・インターフェースは、ローカル・システム上、または、TCP/IP ベースの分散ネットワーク環境のいずれかで、アプリケーション間のプロセス間通信に必要なルーチンを提供します。対等接続が確立された後は、接続を一意的に識別するためにソケット記述子が使用されます。ソケット記述子自体は、タスク固有の数値です。

ソケットによって記述される、TCP/IP ベースの分散ネットワーク・アプリケーションの対等接続の 1 つの終端は、以下のもので一意的に定義されます。

- IP アドレス

  例えば、127.0.0.1 (IPv4 ネットワークの場合) または FF01::101 (IPv6 ネットワークの場合)。
- 通信プロトコル
  - ユーザー・データグラム・プロトコル (User Datagram Protocol (UDP))
  - 伝送制御プロトコル (Transmission Control Protocol (TCP))
- ポート

  アプリケーションを識別する数値。以下のような区別があります。
  - 「ウェルノウン」ポート (例えば、Telnet のポート 23 など)
  - ユーザー定義ポート

ソケット・アプリケーションは、元は Berkeley Software Distribution (BSD) によって定義されたソケット API のバリエーションを使用する、C または C++ アプリケーションであるというのが一般的でした。JAVA 言語でもソケット API を提供しています。JAVA ベースのクライアント/サーバー・アプリケーションでは、これらのソケット・サービスが活用されています。

ソケット・プログラミング・インターフェースは、例えば The Open Group などによって、簡単に移植できるように標準化されています。

UNIX システムでは、TCP/IP ベースのソケットの他に、ローカル UNIX ホスト自体の内部でのプロセス間通信 (IPC) 用のソケット・インターフェースが提供されています。それらの UNIX ソケットは、プロセス間通信にローカル・ファイル・システムを使用します。

z/VSE は、TCP/IP ベースのソケット・サービスを提供します。それらは IPC 用にも使用できますが、ネットワーク通信用にのみ使用することが本来の目的です。

z/VSE で使用可能なソケット・アプリケーション・プログラミング・インターフェース

z/VSE では、一連のさまざまなソケット・アプリケーション・プログラミング・インターフェース (API) を提供しています。これらのインターフェースは、TCP/IP for z/VSE、IPv6/VSE、および Fast Path to Linux 機能 (LFP) でサポートされています。詳しくは、対応する製品情報を参照してください。

- EZA インターフェース
  - z/VSE には、HLASM プログラマー向けの EZASMI マクロ・インターフェースと、COBOL、PL/I、および HLASM のプログラマー向けの EZASOKET 呼び出しインターフェースが用意されています。これらのインターフェースは、対応する z/OS インターフェースとの幅広い互換性を備え、TCP/IP for z/VSE、IPv6/VSE、および LFP でサポートされています。これらのインターフェースの説明については、 229 ページの『第 12 章 CALL 命令アプリケーション・プログラム・インターフェース (EZASOKET API) の使用』と 333 ページの『第 13 章 マクロ・アプリケーション・プログラミング・インターフェース (EZASMI API) の使用』を参照してください。

EZASMI インターフェースでの非同期関数処理は、z/VSE 上でも提供されています。しかし、z/OS と比較すると、ECB メソッドのみが使用可能で、ECB 領域の長さは 160 バイトでなければなりません (z/OS では 104 バイト)。

- Language Environment for z/VSE を使用する TCP/IP API
  - LE/VSE 1.4 C ソケット・インターフェースは、ランタイム環境 (CICS またはバッチ) を動的に判別します。詳しくは、 62 ページの『PL/I』 および 63 ページの『COBOL』 を参照してください。
  - REXX/VSE 内部での REXX/VSE ソケット API サポート。詳しくは、「*REXX/VSE 解説書*」(SC88-6692) を参照してください。
- TCP/IP for z/VSE、ネイティブ API
  - アセンブラー・ソケット・マクロ・インターフェース

    このインターフェースは、ソケット・アプリケーションのコーディングをサポートするだけでなく、TCP/IP 組み込み Telnet、FTP、および LPR アプリケーション・レベル・プロトコルのサポートを使用したリモート・システムへの動的な接続もサポートします。これは、バッチまたは CICS 環境で使用する場合に指定される必要があります。
  - COBOL および PL/I プリプロセッサー・インターフェース

    これは、バッチまたは CICS 環境で使用する場合に指定される必要があります。
  - BSD-C ソケット・インターフェース

    アプリケーションに動的にランタイム環境 (CICS またはバッチ) を判別させることができます。詳しくは、 92 ページの『CICS の考慮事項』を参照してください。
  - REXX ソケット API

    TCP/IP for z/VSE で使用可能な REXX サポートには、次の 2 つのタイプがあります。
    - TCP/IP for z/VSE 内部での REXX サポート (つまり、REXX ソケット API)。この REXX サポートの資料については、「*TCP/IP for VSE Programmer's Guide*」を参照してください。
    - REXX/VSE 内部での REXX/VSE ソケット API サポートについては、「*REXX/VSE 解説書*」(SC88-6692) でもう少し詳しく説明します。

これらのインターフェースについて詳しくは、「*TCP/IP for VSE Programmer's Guide*」を参照してください。

移植性の観点

アセンブラー

TCP/IP for z/VSE アセンブラー・ソケット・マクロ・インターフェースを使用して、プログラムと z/VSE を結合できます。アセンブラー言語には API 標準がないため、ソケット・マクロ・インターフェースを使用する、この言語で作成されたプログラムは、z/VSE オペレーティング・システム以外の環境に移植できません。

多少の違いはありますが、z/OS でも EZASMI マクロ・インターフェースおよび EZASOKET 呼び出しインターフェースを使用できます。これらのインターフェースを使用する z/OS 上のアプリケーションは、簡単に z/VSE に移植することができ、逆方向の移植も同様です。

COBOL および PL/I

COBOL および PL/I は、z/VSE 環境で最も多く使用されているプログラミング言語ですが、TCP/IP ベースのソケット・アプリケーションを作成するための「ネイティブ」言語は C です。特定の環境において、複数のプラットフォームで使用可能な、C 以外の言語用のインターフェースが存在するか、または製品固有のプログラミング・ツールキットを通じてインターフェースが提供されていることもあります。

z/VSE 以外のシステムへの移植性が重要でない場合は、「*TCP/IP for VSE Programmer's Guide*」に記載されている TCP/IP for z/VSE プリプロセッサー API を選択できます。 z/OS または z/VM への移植性が重要である場合は、『Language Environment (言語環境プログラム)』セクションにある詳細説明をお読みください。z/OS への移植性が重要である場合は、EZASOKET 呼び出しインターフェースを使用することを検討してください。

C 言語

上でも説明したように、C は、Java は別として、任意のどのオペレーティング・システム環境でも非常によく似たプログラミング・インターフェースが提供されている、唯一のプログラミング言語です。

C ソケット・インターフェースは、Berkeley Software Distribution (BSD) によって標準化されていますが、他にも、システム間およびプラットフォーム間の移植性を保証する標準があります。例えば、The Open Group の CAE 仕様には、次のような標準があります。「System Interfaces and Headers, Issue 4, Version 2」(XPG4.2 とも呼ばれます)。The Open Group については、インターネットで ***http://www.opengroup.org/*** を参照してください。

Language Environment®

IBM Z プラットフォーム上の Language Environment (LE) は、z/OS、z/VM、および z/VSE 間の移植性を保証します。具体的なニーズと移植性の問題に基づいて、次の言語

- C
- COBOL
- PL/I
- アセンブラー
- REXX

のうちどの言語が、TCP/IP ソケット・インターフェースをベースにしたクライアント/サーバー・アプリケーションの作成に適しているのかが決まります。

LE は、LE 対応の任意の高水準言語 (C、COBOL、PL/I) を使用することによる LE サービスの使用、および、LE 準拠のアセンブラー・プログラムからの LE サービスの使用をサポートします。これには、混合言語アプリケーションのサポートも含まれます。

ソケット・サービスを使用する、C 以外のプログラミング言語で作成された LE ベースのプログラムは、IBM Z サーバーに移植できません。 IBM Z 上の LE は、システム間の互換性を提供します。

LE 対応アプリケーション

アプリケーションが「LE 対応」(または、「LE 準拠」、「LE 規格合致」) と見なされるのは、そのアプリケーションが共通実行環境 (Common Execution Environment (CEE)) モデルに準拠し、そのランタイムのリンケージ、保管、および条件処理モデルに準拠している場合です。これは、LE 準拠のコンパイラーまたは prologue/epilogue マクロを使用して、アプリケーションがコンパイルまたはアセンブルされている場合にも当てはまります。これらは、基本的に、C for VSE、COBOL for VSE、および PL/I for VSE のすべてのコンパイル済みプログラム、および CEEENTRY/CEETERM マクロを使用するすべてのアセンブラー・プログラムです。C prologue/epilogue アセンブラー・マクロを使用するアセンブラー・プログラムを含めて、C/VSE サブルーチンもこの要件を満たします。

どの API を使用するか?

既に説明したとおり、使用に適した言語および API は、次の事項に基づいて選択します。

- 移植性

  クロスプラットフォーム開発が容易に行えます (単一のソース・コード)。

- 互換性

  IBM Z プラットフォームでは、LE プログラミング・インターフェースを使用しているソースについては、z/OS、z/VM、および z/VSE 間で互換性があります。

  LE/VSE では、C 機能テスト・マクロ _XOPEN_SOURCE_EXTENDED によって定義されるインターフェースに焦点が絞られていますが、例えば、z/OS では、機能テスト・マクロ OE_SOCKETS によって有効になる、少しだけ違うインターフェースがそれに追加して提供されています。

- 保守容易性

  ソケット・アプリケーションを TCP/IP 製品から分離することによって、両者を独立して保守 (サービス) できるようになります。

移植性、互換性、および保守容易性の各観点は、選択するプログラミング言語によって現れ方が異なります。

アセンブラー

TCP/IP で提供されている SOCKET マクロは、ソケットをベースにしたアプリケーションの作成をサポートするだけでなく、組み込み Telnet、FTP、および LPR アプリケーション・レベル・プロトコルへのアクセス権限を付与します。
Telnet、FTP、および LPR プロトコルへのアクセスが必要でない場合、LE 準拠のアセンブラー・プログラムは、SOCKET マクロを使用する代わりに LE/VSE C ソケット・インターフェースを呼び出すことで、TCP/IP サービス・レベルからの独立性を獲得できます。

SOCKET マクロに影響する TCP/IP サービスは、アプリケーションの再アセンブルを必要とする場合があります。

EZASMI マクロおよび EZASOKET 呼び出しインターフェースは、対応する z/OS インターフェースとほとんど互換性があります。これによって、クロスプラットフォーム開発が簡単になります。両インターフェースによって、ソケット・アプリケーションは TCP/IP 製品から分離され、その結果、これらの両パーツは別々に保守できるようになります。

COBOL および PL/I

TCP/IP プリプロセッサー API (EXEC TCP ...) を使用して、スタブ・ルーチンがユーザー・アプリケーションとリンク・エディットされます。

- COBOL - IPNETXCO.OBJ
- PL/I - IPNETXP.OBJ

これらのモジュールに影響する TCP/IP サービスは、アプリケーションの再リンクを必要とする場合があります。

プリプロセッサー・インターフェースの使用法の例を以下に示します。

```
*
*    Attempt to open a connection
*
 EXEC TCP OPEN FOREIGNPORT(2000)
               FOREIGNIP(IPADDRESS)
               LOCALPORT(0)
               RESULTAREA(RESULTS)
               DESCRIPTOR(MY-DESC)
               ACTIVE
               WAIT(YES)
               ERROR(SECOND-TEST)
 END-EXEC.
```

EZASOKET 呼び出しインターフェースは COBOL for VSE および PL/I for VSE プログラムでも使用できることに注意してください。

C 言語

TCP/IP 環境での主要な言語は C であることから、LE/VSE には C ソケット・インターフェースしかありません。ただし、LE/VSE と、z/OS および z/VM の Language Environment により、LE サービスはアセンブラー、COBOL、および PL/I からも呼び出すことができます。また、COBOL および PL/1 プログラムから EZASOKET インターフェースを使用することもできます。

LE/VSE C ベースのソケット・アプリケーションの論理的な制御フローは、次の図のように表されます。LE/C ランタイムは、アプリケーションを特定の TCP/IP 製品から分離します。LE/C TCP/IP ソケット API マルチプレクサーでは、実行時に適切な TCP/IP スタックを選択できます。TCP/IP スタック・パーティションの処理では、デフォルトとして $EDCTCPV.PHASE が使用されます。その他の LE/C TCP/IP インターフェース・ルーチンを使用する場合は、LE/C TCP/IP ソケット API マルチプレクサーを構成できます。例えば、LFP LE/C TCP/IP インターフェースを使用する場合はフェーズ IJBLFPLE、IPv6/VSE インターフェースを使用する場合はフェーズ フェーズ BSTTTCP6 です。マルチプレクサーを構成するには、ICCF ライブラリー 62 にあるスケルトン EDCTCPMC を使用します。LE/C ソケット API マルチプレクサーを使用して、アプリケーションで使用する TCP/IP および SSL の実装を選択する方法については、95 ページの『第 10 章 使用する TCP/IP と SSL の実装を選択』を参照してください。

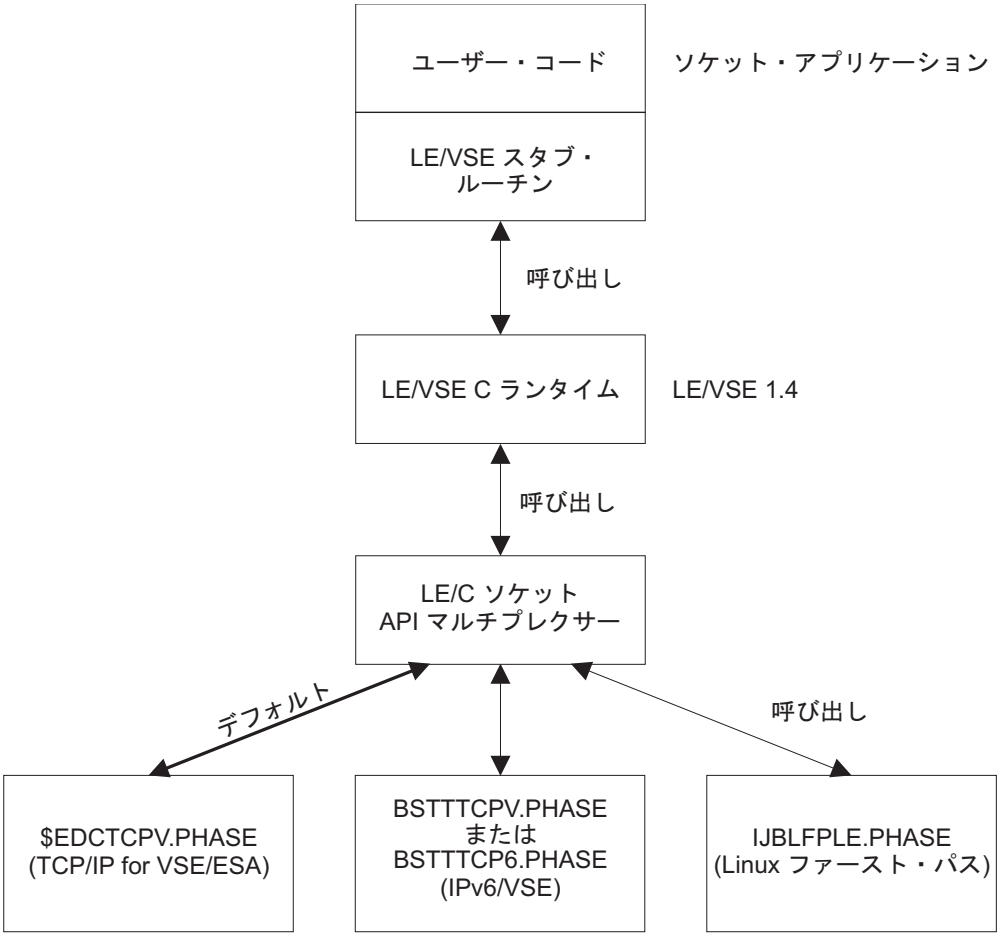


図 *12.* 各種 *TCP/IP* スタックで *LE/VSE C* ソケットを使用する際の制御フロー。

注**:**

1. C for VSE コンパイラーを使用する場合は、Language Environment 1.4 で提供されるソケット API を使用してください。必要な C ヘッダー・ファイルは、VSE ライブラリー PRD2.SCEEBASE に入っています。
2. 非 LE 対応の C コンパイラー、例えば、C/370™ を使用する場合は、ネイティブ TCP/IP for z/VSE BSD-C インターフェースを使用するように制限されて

います。これには、VSE ライブラリー PRD2.TCPIPC に入って出荷される **socket.h** インクルード・ファイルの使用が含まれます。

LE/C ソケット・インターフェースは、TCP/IP for z/VSE、IPv6/VSE、および Linux ファスト・パスで使用可能です。詳しくは、対応する製品情報を参照してください。

LE/VSE ソケット API の活用

LE ランタイム・サービスを使用するアプリケーション (C、COBOL、および PL/I)、または LE 対応アセンブラー・プログラムは、直接 (C) または LE 言語間通信 (Interlanguage Communication (ILC)) サポートを使用して、LE/VSE C ソケット・ルーチンを使用できます。また、COBOL および PL/1 プログラムから EZASOKET インターフェースを使用することもできます。

C 言語

LE/VSE は、C 用のソケット・プログラミング・インターフェースを提供しています。詳しくは、 101 ページの『第 11 章 TCP/IP による LE/VSE C ソケット・インターフェースのサポート』を参照してください。デフォルトでは、これらのインターフェースは、TCP/IP スタック・パーティションを処理する際に $EDCTCPV.PHASE を使用します。ご使用の VSE システムに TCP/IP for z/VSE がインストールされている場合、$EDCTCPV.PHASE は TCP/IP for z/VSE ライブラリーに入っています。その他の LE/C TCP/IP インターフェース・ルーチンを使用する場合は、LE/C TCP/IP ソケット API マルチプレクサーを構成できます。例えば、LFP LE/C TCP/IP インターフェースを使用する場合はフェーズ IJBLFPLE、IPv6/VSE インターフェースを使用する場合はフェーズ フェーズ BSTTTCP6 です。マルチプレクサーを構成するには、ICCF ライブラリー 62 にあるスケルトン EDCTCPMC を使用します。

TCP/IP HLL インターフェースには、基本的に、
OPEN、SEND、RECEIVE、CLOSE インターフェースがあるだけですが、C 言語呼び出しではもっときめ細かくなっています。必要な呼び出しは、サーバーとクライアントのどちらのプログラムを書くのかによって違います。

クライアント

クライアント・アプリケーションの簡単なコード・ロジックの例を以下に示します。

```
socket()           - create a socket
   ↓
connect()          - bind and connect to server
   ↓
send() / receive() - data interchange
   ↓
close()            - destroy socket
```

サーバー

サーバー (デーモン) アプリケーションの簡単なコード・ロジックの例を以下に示します。

```
socket()           - create a socket using a specific protocol
   ↓
bind()             - bind the socket to a port
   ↓
listen()           - make it a passive socket
   ↓
[ select() ]       - wait for incoming connections
   ↓
accept()           - connect to caller
   ↓
getsockname()      - determine caller
   ↓
send() / receive() - data interchange
   ↓
close()            - destroy socket
```

この例で示した、大括弧で囲んだ select() 呼び出しを使用して、複数のクライアントを同時に操作できます。これは、一連のソケットでアクティビティーを待機するのに使用され、WAITM (wait multiple) オペレーティング・システム呼び出しに似ています。従って、サーバー・アプリケーションは、新規クライアントが接続するのを待機 (accept() 呼び出し) するのと同時に、既に接続済みのクライアントからの要求を待機 (receive() 呼び出し) できます。

アセンブラー言語

LE/VSE は、アセンブラー・プログラムからの C サブルーチンの呼び出しをサポートします。

アセンブラー・ソース

以下の例に示すコード断片は、LE マクロ CEEENTRY を使用して、Language Environment を使用可能にしています。次に、TCP/IP サブルーチン GETHNAM を呼び出しています。最後に、このルーチンは CEETERM を呼び出して、もう必要のない Language Environment を使用不可にしています。

注**:** LE は、アプリケーションの先頭で使用可能にし、最後に終了することをお勧めします。LE の開始/終了による高いオーバーヘッドを避けるため、このシーケンスを必要以上に呼び出さないでください。

```
*
GETHOSTN  CEEENTRY PPA=MAINPPA,MAIN=YES
*
*
          LA     1,PARMSTR
          CALL   GETHNAM
*
          LTR    15,15
          BZ     RETOK
          WTO    'GETHOSTNAME() FAILED'
          B      RTNEND
RETOK     WTO    'GETHOSTNAME() SUCCESSFUL'
*
RTNEND    CEETERM
*
CBUFLEN    EQU  20
PARMSTR   DC   A(HNAME)
          DC   F(CBUFLEN)
HNAME     DS   CL(CBUFLEN)
```

アセンブラーから呼び出される、OS リンケージを使用する C サブルーチン

次の例では、C ルーチン gethostname() を呼び出す OS リンケージ規約を使用する、スタブ・ルーチンのコーディング方法を示します。

```
#include <types.h>
#include <unistd.h>

#pragma linkage(GETHOSTNAME, OS)
#pragma map(GETHOSTNAME, GETHNAM)

int GETHOSTNAME( char   *buffer,
                 size_t  size)
{

   return( gethostname( buffer, size));
}
```

PL/I

C と PL/I 間の LE/VSE 言語間通信 (ILC) は、PL/I for VSE/ESA に対してのみ提供されています。

マニュアル「*Writing Interlanguage Communication Applications*」(SC33-6686) に、ILC 呼び出しの使用方法の詳しい解説があります。

アセンブラーの例と同様に、PL/I リンケージを伴う C スタブ・ルーチンがなければなりません。以下の点に注意してください。

- C での NULL は x'00000000'、PL/I での NULL は x'FF000000' です。従って、PL/I プログラムは、必要な箇所では SYSNULL (x'00000000') をチェックする必要があります。
- C の場合、文字ストリングは x'00' 終端標識 (最終バイト) で論理的にアンバインドされます。

従って、gethostname() を呼び出すためのスタブ・ルーチンは、次のようになります。

```
#include <types.h>
#include <unistd.h>

#pragma linkage(GETHOSTNAME, PLI)
#pragma map(GETHOSTNAME, GETHNAM)

int GETHOSTNAME( char   **buffer,
                 size_t   size)
{

   return( gethostname( *buffer, size));

}
```

これに対応する PL/I のコード断片は、次のようにサブルーチンを呼び出します。

```
...
DCL GETHNAM EXTERNAL ENTRY
    RETURNS(FIXED BIN(31));
DCL HOSTNAME CHAR(20);
DCL HNSIZE FIXED BIN(31);
DCL CRC FIXED BIN(31);
```

```
...
HNSIZE = 20;
CRC = GETHNAM(ADDR(HOSTNAME),(HNSIZE));
...
```

COBOL

C と COBOL 間の LE/VSE 言語間通信 (ILC) は、COBOL for VSE/ESA リリース 1 に対して提供されています。

マニュアル「*Writing Interlanguage Communication Applications*」(SC33-6686) に、ILC 呼び出しの使用方法の詳しい解説があります。

LE C ルーチン gethostname() を呼び出してローカル・ホスト名を取得する方法の例を以下に示します。

```
 IDENTIFICATION DIVISION.

         PROGRAM-ID.      C2COB2.
         AUTHOR.          INGO ADLUNG.
           INSTALLATION.  BOEBLINGEN GERMANY.
           DATE-WRITTEN.  MAY 19, 1999.
         DATE-COMPILED.

        ENVIRONMENT DIVISION.
         CONFIGURATION SECTION.
          SOURCE-COMPUTER.   IBM-370.
          OBJECT-COMPUTER.   IBM-370.

       DATA DIVISION.
        WORKING-STORAGE SECTION.
        01  RESULTS.
            05 RVALUE       PIC S9(9) BINARY.
            05 RDETAIL      PIC S9(9) BINARY.
        01  BUFSIZE     PIC S9(9) BINARY.
        01  BUFFER.
            05  WORKAREA    PICTURE X(64).

      PROCEDURE DIVISION.

      MAIN.
     *
     *  Display the name of the host we are running on
     *
          MOVE 64 TO BUFSIZE.

          DISPLAY 'Calling C gethostname()' UPON CONSOLE.

          CALL 'COBGHNAM' USING BY REFERENCE WORKAREA
                                BY CONTENT BUFSIZE
                                BY REFERENCE RVALUE, RDETAIL.

          DISPLAY WORKAREA UPON CONSOLE.

          STOP RUN.
```

gethostname() を COBOL リンケージを使用して呼び出すための、対応する C スタブ・ルーチンは、次のようになります。

```
#include <types.h>
#include <unistd.h>
#pragma linkage(cobol_gethostname, COBOL)
#pragma map(cobol_gethostname, COBGHNAM)
```

```
void cobol_gethostname( char   *buffer,
                  size_t  size,
                  int    *return)
{

   *return = gethostname( buffer, size);

}
```

COBOL での LE C ソケット・サービスの使用例

次の例は、言語間通信アプリケーションをコーディングするための LE の機能に基づいています。

ソース・コード全体は、z/VSE ホーム・ページ http://www.ibm.com/systems/z/os/zvse/downloads/samples.html で FTP ダウンロード用リンクから cobsock.zip として入手できます。

下の図に、ごく基本的なサーバー・アプリケーションの COBOL ソース・コードを示します。複雑さを軽減するため、この例では単一のクライアントのみが処理され、通信障害が発生した場合に必要なエラー・リカバリーは含まれていません。

```
IDENTIFICATION DIVISION.

         PROGRAM-ID.      C2COB1.
         AUTHOR.          INGO ADLUNG.
           INSTALLATION.  BOEBLINGEN GERMANY.
           DATE-WRITTEN.  MAY 4, 1998.
         DATE-COMPILED.

        ENVIRONMENT DIVISION.
         CONFIGURATION SECTION.
          SOURCE-COMPUTER.   IBM-370.
          OBJECT-COMPUTER.   IBM-370.

        DATA DIVISION.
         WORKING-STORAGE SECTION.
         01  SOCKET-DATA.
             05  DOMAIN      PIC S9(9) BINARY.
             05  SOCKTYPE    PIC S9(9) BINARY.
             05  PROTOCOL    PIC S9(9) BINARY.
             05  LSOCKET     PIC S9(9) BINARY.
             05  RSOCKET     PIC S9(9) BINARY.
         01  SOCKADDR-IN.
             05  SIN-FAMILY  PIC S9(2) BINARY.
             05  SIN-PORT    PICTURE S9(4) BINARY.
             05  SIN-ADDR    PIC S9(9) BINARY.
             05  SIN-ZERO    PIC S9(2) BINARY OCCURS 4 TIMES VALUE 0.
         01  RESULTS.
             05 RVALUE       PIC S9(9) BINARY.
             05 RDETAIL      PIC S9(9) BINARY.
         01  BUFSIZE     PIC S9(9) BINARY.
         01  L-COUNT     PIC S9(9) BINARY.
         01  BUFFER.
             05  WORKAREA    PICTURE X(512).

       PROCEDURE DIVISION.

       MAIN.

      *
      *  Create a TCP stream socket. The socket value will be
      *   returned in variable RVALUE.
      *
```

```
*   domain type AF_INET is 2
*   socket type SOCK_STREAM is 1
*   protocol IPPROTO_TCP is 6
*
     MOVE 2 TO DOMAIN.
     MOVE 1 TO SOCKTYPE.
     MOVE 6 TO PROTOCOL.

     DISPLAY 'Calling C socket()'.

     CALL 'TCPSOCKT' USING BY CONTENT DOMAIN, SOCKTYPE, PROTOCOL
                           BY REFERENCE RVALUE, RDETAIL.

     MOVE RVALUE TO LSOCKET.

*
*  Bind the socket to the local port
*
*   domain type AF_INET is 2
*   local port is 2000
*
     MOVE 2    TO SIN-FAMILY.
     MOVE 2000 TO SIN-PORT.
     MOVE 0    TO SIN-ADDR.
     MOVE 16   TO BUFSIZE.

     DISPLAY 'Calling C bind()'.

     CALL 'TCPBIND' USING BY CONTENT LSOCKET
                          BY REFERENCE SOCKADDR-IN
                          BY CONTENT BUFSIZE
                          BY REFERENCE RVALUE, RDETAIL.

*
*  Convert socket to passive mode.
*
     MOVE 1 TO L-COUNT.

     DISPLAY 'Calling C listen()'.

     CALL 'TCPLIST' USING BY CONTENT LSOCKET, L-COUNT
                          BY REFERENCE RVALUE, RDETAIL.

*
*  Wait for incoming clients.
*
     INITIALIZE SOCKADDR-IN.
     MOVE 16 TO BUFSIZE.

     DISPLAY 'Calling C accept()'.

     CALL 'TCPACCP' USING BY CONTENT LSOCKET,
                          BY REFERENCE SOCKADDR-IN, BUFSIZE,
                                       RVALUE, RDETAIL.

*
*  Receive a piece of data.
*
     MOVE RVALUE TO RSOCKET.
     MOVE 512 TO BUFSIZE.

     DISPLAY 'Calling C read()'.

     CALL 'TCPREAD' USING BY CONTENT RSOCKET
                          BY REFERENCE WORKAREA
                          BY CONTENT BUFSIZE
                          BY REFERENCE RVALUE, RDETAIL.
```

```
*
      *  Send the data back to the caller
      *
           MOVE RVALUE TO BUFSIZE.

           DISPLAY 'Calling C write()'.

           CALL 'TCPWRITE' USING BY CONTENT RSOCKET
                                 BY REFERENCE WORKAREA
                                 BY CONTENT BUFSIZE
                                 BY REFERENCE RVALUE, RDETAIL.
      *
      *  Close the connection
      *
           DISPLAY 'Calling C close()'.

           CALL 'TCPCLOSE' USING BY CONTENT RSOCKET
                                 BY REFERENCE RVALUE, RDETAIL.

      *
      *  Release the listen socket too.
      *
           DISPLAY 'Calling C close()'.

           CALL 'TCPCLOSE' USING BY CONTENT LSOCKET
                                 BY REFERENCE RVALUE, RDETAIL.


           STOP RUN.
```

次の例に示すのは、対応する C ソースで、ソケット・ルーチンのマッピングを提供します。生成されるオブジェクト・デックは、生成された COBOL オブジェクト・デックとリンク・エディットする必要があります。

```
#include <types.h>
#include <unistd.h>
#include <in.h>
#include <socket.h>
#include <errno.h>
#include <stdio.h>

#pragma linkage( cob2c_socket, COBOL)
#pragma linkage( cob2c_bind,   COBOL)
#pragma linkage( cob2c_listen, COBOL)
#pragma linkage( cob2c_accept, COBOL)
#pragma linkage( cob2c_read  , COBOL)
#pragma linkage( cob2c_write,  COBOL)
#pragma linkage( cob2c_close,  COBOL)


#pragma map( cob2c_socket, "TCPSOCKT")
#pragma map( cob2c_bind,   "TCPBIND" )
#pragma map( cob2c_listen, "TCPLIST" )
#pragma map( cob2c_accept, "TCPACCP" )
#pragma map( cob2c_read,   "TCPREAD" )
#pragma map( cob2c_write,  "TCPWRITE")
#pragma map( cob2c_close,  "TCPCLOSE")

void cob2c_socket( int  domain,
                   int  type,
                   int  protocol,
                   int *psocket,
                   int *perr)
{
```

```
     printf(
       "socket() called, domain : %d, type : %d, protocol : %d¥n",
       domain, type, protocol);

     *psocket = socket( domain, type, protocol);
     *perr    = errno;
}

void cob2c_bind( int                    socket,
                 const struct sockaddr *address,
                 size_t                 len,
                 int                   *pvalue,
                 int                   *perr)
{
     struct sockaddr_in * sockin = (struct sockaddr_in *)address;

     *pvalue = bind( socket, address, len);
     *perr   = errno;
}

void cob2c_listen( int  socket,
                   int  backlog,
                   int *pvalue,
                   int *perr)
{
     *pvalue = listen( socket, backlog);
     *perr   = errno;
}

void cob2c_accept( int              socket,
                   struct sockaddr *address,
                   size_t          *len,
                   int             *pvalue,
                   int             *perr)
{
     *pvalue = accept( socket, address, len);
     *perr   = errno;
}

void cob2c_read( int     socket,
                 void   *buffer,
                 size_t  len,
                 size_t *pvalue,
                 int    *perr)
{
     *pvalue = read( socket, buffer, len);
     *perr   = errno;
}

void cob2c_write( int         socket,
                  const void *buffer,
                  size_t      len,
                  size_t     *pvalue,
                  int        *perr)
{
     *pvalue = write( socket, buffer, len);
     *perr   = errno;
}

void cob2c_close( int         socket,
                  size_t     *pvalue,
                  int        *perr)
{
     *pvalue = close( socket);
     *perr   = errno;
}
```

EZASMI/EZASOKET プログラミング・インターフェースの活用

z/VSE 上のアプリケーションは、EZASMI または EZASOKET、あるいはその両方のプログラミング・インターフェースを使用できます。これらのプログラミング・インターフェースは、バッチ環境と CICS Transaction Server 環境の両方で、プログラミング用に提供されています。

以下のいくつかのサンプル・プログラムで、これらのインターフェースのシンプルな使用法を示します。複雑さを軽減するため、通信障害が発生した場合に必要なエラー・リカバリーは含まれていません。最初のサンプルは、EZASMI マクロ・インターフェースを使用するクライアント・アセンブラー・プログラムの例です。

図 *13. EZASMI* マクロを *(*同期して*)* 使用するサンプル・プログラム

```
*        PRINT NOGEN
*********************************************************************
*                                                                   *
*   MODULE NAME:  SAMPCLIE                                          *
*                                                                   *
*   FUNCTION: Sample program for usage of EZASMI macro              *
*             (Client part)                                         *
*                                                                   *
*   ATTRIBUTES: NON-REUSABLE                                        *
*                                                                   *
*   REGISTER USAGE:                                                 *
*        R3  = BASE REG                                             *
*        R13 = SAVE AREA                                            *
*                                                                   *
*   INPUT: NONE                                                     *
*   OUTPUT: NONE                                                    *
*                                                                   *
*********************************************************************
---------------------------------------------------------------------*
* START OF EXECUTABLE CODE                                          *
*-------------------------------------------------------------------*
SAMPCLIE START X'78'                  adjust addr behind part savearea
SAMPCLIE AMODE ANY
SAMPCLIE RMODE ANY
         USING *,R15                  Use Entry Register for base
         B     SAMPCLST
         DC    C'SAMPCLST-00/06/23'
*
SAMPCLST DS    0H
         STM   R14,R12,12(R13)        Save Caller's Registers
         LR    R3,R15                 Change base register to R3
         DROP  R15                    Done with this register
         USING SAMPCLIE,R3            Tell assembler about new base
         LA    R15,MYSAVE             Get addr of own save area
         ST    R13,MYSAVE+4           Save caller's save area addr
         ST    R15,8(R13)             Save own save area addr
         LR    R13,R15                Load Reg13
*********************************************************************
*                                                                   *
*        Issue INITAPI to connect to interface                      *
*********************************************************************
         EZASMI TYPE=INITAPI,         Issue INITAPI Macro             X
               MAXSOC=MAXSOC,         Max number of sockets (in)      X
               MAXSNO=MAXSNO,         Greatest Descr Number used (out)X
               ERRNO=ERRNO,           ERRNO field                     X
               RETCODE=RETCODE        RETCODE field
*
*********************************************************************
```

```
*        Issue SOCKET call                                              *
*********************************************************************
         EZASMI TYPE=SOCKET,           Issue SOCKET call               X
               AF='INET',              INTERNET family                 X
               SOCTYPE='STREAM',       Stream socket                   X
               PROTO=PROTOCOL,         protocol                        X
               ERRNO=ERRNO,            ERRNO field                     X
               RETCODE=RETCODE         RETCODE field
         MVC   SOCKET1,RETCODE         Save the socket descriptor
*
*********************************************************************
*        Issue CONNECT                                                  *
*********************************************************************
         EZASMI TYPE=CONNECT,          Issue CONNECT call              X
               S=SOCKET1+2,            socket descriptor (halfword)    X
               NAME=SAMPSERV,          to SAMPSERV program             X
               ERRNO=ERRNO,            ERRNO field                     X
               RETCODE=RETCODE         RETCODE field
*
*********************************************************************
*        Issue WRITE on connected socket                                *
*********************************************************************
         EZASMI TYPE=WRITE,            Issue WRITE  call               X
               S=SOCKET1+2,            on this socket                  X
               NBYTE=MSG1L,            Length of first message         X
               BUF=MSG1,               Text of first message           X
               ERRNO=ERRNO,            ERRNO field                     X
               RETCODE=RETCODE         RETCODE field
         B     READ1                   go and read
*
MSG1L    DC    F'40'
MSG1     DC    CL40'DATA SENT FROM SAMPCLIE.'
*********************************************************************
*        Issue READ on connected socket                                 *
*********************************************************************
READ1    EZASMI TYPE=READ,             Issue READ   call               X
               S=SOCKET1+2,            on this socket                  X
               NBYTE=READBL,           length of read buffer           X
               BUF=READB,              address of read buffer          X
               ERRNO=ERRNO,            ERRNO field                     X
               RETCODE=RETCODE         RETCODE field
*
*********************************************************************
*        Issue CLOSE     on connected socket                            *
*********************************************************************
         EZASMI TYPE=CLOSE,            Issue CLOSE  call               X
               S=SOCKET1+2,            on this socket                  X
               ERRNO=ERRNO,            ERRNO field                     X
               RETCODE=RETCODE         RETCODE field
*
*********************************************************************
*                                                                      *
*        Issue TERMAPI to disconnect interface                          *
*********************************************************************
         EZASMI TYPE=TERMAPI           Issue TERMAPI call
*
         EOJ
         EJECT
*--------------------------------------------------------------------*
* CONSTANTS/VARIABLES USED BY THIS PROGRAM                            *
*--------------------------------------------------------------------*
         EZASMI TYPE=TASK,STORAGE=CSECT Task Storage Area
MYSAVE   DC    18F'0'                  Register Save Area
ERRNO    DC    F'0'
RETCODE  DC    F'0'
*--------------------------------------------------------------------*
* INITAPI macro parms *
```

```
*--------------------*
MAXSOC   DC    H'256'                  MAXSOC parm value
MAXSNO   DC    F'0'                    Highest socket descriptor avail
*-------------------------------------------------------------------*
* SOCKET macro parms *
*--------------------*
PROTOCOL DC    F'0'                    default protocol
SOCKET1  DC    F'0'                    save area for socket descriptor
*
*-------------------------------------------------------------------*
* CONNECT Macro Parms*
*--------------------*
         CNOP  0,4
SAMPSERV DC    0CL16' '                SOCKET NAME structure of SERVER
         DC    AL2(2)                  FAMILY (AF-INET)
         DC    H'4000'                 Port of SAMPSERV
         DC    AL1(9),AL1(164),AL1(155),AL1(122) IP-Addr of SAMPSERV
         DC    XL8'00'                 RESERVED
*
*-------------------------------------------------------------------*
* READ MACRO PARMS   *
*--------------------*
READBL   DC    F'40'                   SIZE OF READ BUFFER
READB    DC    CL40' '                 READ BUFFER
**----  register equates ------------------------------------------*
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*
         END   SAMPCLIE
```

2 番目のサンプルは、非同期の EZASMI マクロ・インターフェースを使用するサーバー・アセンブラー・プログラムの例を示します。

図 *14. EZASMI* マクロを *(*非同期に*)* 使用するサンプル・プログラム

```
*        PRINT NOGEN
*********************************************************************
*                                                                   *
*   MODULE NAME:  SAMPSERV                                          *
*                                                                   *
*   FUNCTION: Sample Program for EZASMI (asynchronous) macro usage  *
*             (Server Part)                                         *
*                                                                   *
*   ATTRIBUTES: NON-REUSABLE                                        *
*               NON-LE Enabled                                      *
*                                                                   *
*   REGISTER USAGE:                                                 *
*        R3  = BASE REG 1                                           *
*        R13 = SAVE AREA                                            *
*                                                                   *
*   INPUT: NONE                                                     *
```

```
*   OUTPUT: NONE                                                        *
*                                                                       *
*************************************************************************
*-----------------------------------------------------------------------*
* START OF EXECUTABLE CODE                                              *
*-----------------------------------------------------------------------*
SAMPSERV START X'78'                  adjust addr behind part savearea
SAMPSERV AMODE 31
SAMPSERV RMODE ANY
         USING *,R15                  Use Entry Register for base
         B     SAMPSTRT
         DC    C'SAMPSEST-00/06/23'
*
SAMPSTRT DS    0H
         STM   R14,R12,12(R13)        Save Caller's Registers
         LR    R3,R15                 Change base register to R3
         DROP  R15                    Done with this register
         USING SAMPSERV,R3            Tell assembler about new base
         LA    R15,MYSAVE             Get addr of own save area
         ST    R13,MYSAVE+4           Save caller's save area addr
         ST    R15,8(R13)             Save own save area addr
         LR    R13,R15                Load Reg13
*************************************************************************
*        Issue INITAPI to connect to interface                          *
*************************************************************************
         EZASMI TYPE=INITAPI,         Issue INITAPI Macro             X
               MAXSOC=MAXSOC,         Max number of sockets (in)      X
               MAXSNO=MAXSNO,         Greatest Descr Number used (out)X
               ASYNC='ECB',           asynchronous ECB processing     X
               ERRNO=ERRNO,           ERRNO field                     X
               RETCODE=RETCODE        RETCODE field
*
*************************************************************************
*        Issue SOCKET call                                              *
*************************************************************************
         XC    ECB,ECB
         EZASMI TYPE=SOCKET,          Issue SOCKET call               X
               AF='INET',             INTERNET family                 X
               SOCTYPE='STREAM',      Stream socket                   X
               PROTO=PROTOCOL,        protocol                        X
               ECB=*ECBA,             wait on this ECB                X
               ERRNO=ERRNO,           ERRNO field                     X
               RETCODE=RETCODE        RETCODE field
*
         WAIT  ECB                    Wait on ECB
         MVC   SOCKET1,RETCODE        Save the socket descriptor
*************************************************************************
*        Issue BIND call                                                *
*************************************************************************
         XC    ECB,ECB                Clear ECB
         EZASMI TYPE=BIND,            Issue BIND   call               X
               S=SOCKET1+2,           socket descriptor               X
               NAME=MYNAME,           Name structure                  X
               ECB=*ECBA,             wait on this ECB                X
               ERRNO=ERRNO,           ERRNO field                     X
               RETCODE=RETCODE        RETCODE field
*
         WAIT  ECB                    Wait on ECB
*************************************************************************
*        Issue LISTEN                                                   *
*************************************************************************
         XC    ECB,ECB                Clear ECB
         EZASMI TYPE=LISTEN,          Issue LISTEN call               X
               S=SOCKET1+2,           socket descriptor               X
               BACKLOG=BACKLOG,       max number of backlog msgs      X
               ECB=*ECBA,             wait on this ECB                X
               ERRNO=ERRNO,           ERRNO field                     X
```

```
               RETCODE=RETCODE         RETCODE field
*
         WAIT  ECB                     Wait on ECB
*********************************************************************
*        Issue ACCEPT                                               *
*********************************************************************
         XC    ECB,ECB                 Clear ECB
         EZASMI TYPE=ACCEPT,           Issue ACCEPT call              X
               S=SOCKET1+2,            socket descriptor              X
               NAME=NAMECLIE,          Name structure of client       X
               ECB=*ECBA,              wait on this ECB               X
               ERRNO=ERRNO,            ERRNO field                    X
               RETCODE=RETCODE         RETCODE field
*
         WAIT  ECB                     Wait on ECB
         MVC   SOCKETN,RETCODE         Save RETCODE (New Socket Descr.)
*********************************************************************
*        Issue READ                                                 *
*********************************************************************
         XC    ECB,ECB                 Clear ECB
         EZASMI TYPE=READ,             Issue READ call                X
               S=SOCKETN+2,            on this socket                 X
               NBYTE=READBUFL,         length of read buffer          X
               BUF=READBUF,            address of read buffer         X
               ECB=*ECBA,              wait on this ECB               X
               ERRNO=ERRNO,            ERRNO field                    X
               RETCODE=RETCODE         RETCODE field
*
         WAIT  ECB                     Wait on ECB
*********************************************************************
*        Issue WRITE     on connected socket                        *
*********************************************************************
         XC    ECB,ECB                 Clear ECB
         EZASMI TYPE=WRITE,            Issue WRITE  call              X
               S=SOCKETN+2,            on this socket                 X
               NBYTE=MSGL,             Length of first message        X
               BUF=MSG,                Text of first message          X
               ECB=*ECBA,              wait on this ECB               X
               ERRNO=ERRNO,            ERRNO field                    X
               RETCODE=RETCODE         RETCODE field
*
 WAIT  ECB                    Wait on ECB
         B     CLOSE1
*
MSGL     DC    F'40'
MSG      DC    CL40'SAMPSERV RECEIVED YOUR DATA.'
*********************************************************************
*        Issue CLOSE socket                                         *
*********************************************************************
CLOSE1   XC    ECB,ECB                 Clear ECB
         EZASMI TYPE=CLOSE,            Issue CLOSE  call              X
               S=SOCKETN+2,            on this socket                 X
               ECB=*ECBA,              wait on this ECB               X
               ERRNO=ERRNO,            ERRNO field                    X
               RETCODE=RETCODE         RETCODE field
*
         WAIT  ECB                     Wait on ECB
*********************************************************************
*        Issue CLOSE socket                                         *
*********************************************************************
         XC    ECB,ECB                 Clear ECB
         EZASMI TYPE=CLOSE,            Issue CLOSE call               X
               S=SOCKET1+2,            on this socket                 X
               ECB=*ECBA,              wait on this ECB               X
               ERRNO=ERRNO,            ERRNO field                    X
               RETCODE=RETCODE         RETCODE field
*
```

```
         WAIT  ECB                     Wait on ECB
*********************************************************************
*        Issue TERMAPI to disconnect interface                      *
*********************************************************************
         EZASMI TYPE=TERMAPI           Issue TERMAPI Call
         EOJ
         EJECT
*-------------------------------------------------------------------*
* CONSTANTS/VARIABLES USED BY THIS PROGRAM                          *
*-------------------------------------------------------------------*
         EZASMI TYPE=TASK,STORAGE=CSECT Task Storage Area
MYSAVE   DC    18F'0'                  Register Save Area
ERRNO    DC    F'0'
RETCODE  DC    F'0'
ECBA     DC    A(ECB)                  POINTER to ECB
ECB      DC    F'0'                    ECB
ECBX     DC    XL156'00'               ECB Extension Area
*
*-------------------------------------------------------------------*
* INITAPI macro parms *
*---------------------*
MAXSOC   DC    H'80'                   MAXSOC PARM VALUE
MAXSNO   DC    F'0'                    Highest Socket Descriptor avail
*
*-------------------------------------------------------------------*
* SOCKET macro parms *
*--------------------*
PROTOCOL DC    F'0'                    default protocol
SOCKET1  DC    F'0'                    savearea for socket descriptor
SOCKETN  DC    F'0'                    savearea for socket descriptor
*
*-------------------------------------------------------------------*
* BIND MACRO PARMS   *
*--------------------*
         CNOP  0,4
MYNAME   DC    0CL16' '                SOCKET NAME STRUCTURE
         DC    AL2(2)                  FAMILY (AF-INET)
MYPORT   DC    H'4000'                 bind to this port
MYADDR   DC    AL1(9),AL1(164),AL1(155),AL1(122)  and IP address
         DC    XL8'00'                 RESERVED
*-------------------------------------------------------------------*
* LISTEN PARMS       *
*--------------------*
BACKLOG  DC    F'5'                    BACKLOG
*-------------------------------------------------------------------*
* ACCEPT PARMS       *
*--------------------*
NAMECLIE DC    0CL16' '                SOCKET NAME STRUCTURE of client
         DC    AL2(2)                  FAMILY
PORTCLIE DC    H'0'
ADDRCLIE DC    F'0'
         DC    XL8'00'                 RESERVED
*-------------------------------------------------------------------*
* READ MACRO PARMS   *
*--------------------*
READBUFL DC    F'40'                   SIZE OF READ BUFFER
READBUF  DC    CL40'none'              READ BUFFER
* ------ register equates ------------------------------------------
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
```

```
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*
         END   SAMPSERV
```

もちろん、この単純なプログラムが非同期インターフェースを使用することが本当に必要なわけではありません。ソケット呼び出しの完了を待つ間にプログラムで他のタスクを実行する必要がある場合は、非同期処理が役立ちます。

次のサンプルは、EZASOKET インターフェースを使用する、COBOL で書かれた同様のサーバー・プログラムです。

図 *15. EZASOKET* 呼び出しを使用する *COBOL* サンプル・プログラム

```
 CBL LIB APOST RMODE(ANY)                                                SAM00010
       IDENTIFICATION DIVISION.                                          SAM00020
                                                                         SAM00030
       PROGRAM-ID.        SAMPSERV                                       SAM00040
       AUTHOR.            HEINZ HAGEDORN                                 SAM00050
           INSTALLATION.  HIER.                                          SAM00060
           DATE-WRITTEN.  June 23, 2000                                  SAM00070
       DATE-COMPILED.                                                    SAM00080
                                                                         SAM00090
       ENVIRONMENT DIVISION.                                             SAM00100
                                                                         SAM00110
       CONFIGURATION SECTION.                                            SAM00120
                                                                         SAM00130
       SOURCE-COMPUTER.   IBM-370.                                       SAM00140
       OBJECT-COMPUTER.   IBM-370.                                       SAM00150
                                                                         SAM00160
       DATA DIVISION.                                                    SAM00170
                                                                         SAM00180
                                                                         SAM00190
       WORKING-STORAGE SECTION.                                          SAM00200
       01 SOKET-FUNCTIONS.                                               SAM00210
          02 SOKET-ACCEPT        PIC X(16) VALUE 'ACCEPT          '.    SAM00220
          02 SOKET-BIND          PIC X(16) VALUE 'BIND            '.    SAM00230
          02 SOKET-CLOSE         PIC X(16) VALUE 'CLOSE           '.    SAM00240
          02 SOKET-CONNECT       PIC X(16) VALUE 'CONNECT         '.    SAM00250
          02 SOKET-INITAPI       PIC X(16) VALUE 'INITAPI         '.    SAM00260
          02 SOKET-LISTEN        PIC X(16) VALUE 'LISTEN          '.    SAM00270
          02 SOKET-READ          PIC X(16) VALUE 'READ            '.    SAM00280
          02 SOKET-SOCKET        PIC X(16) VALUE 'SOCKET          '.    SAM00290
          02 SOKET-TERMAPI       PIC X(16) VALUE 'TERMAPI         '.    SAM00300
          02 SOKET-WRITE         PIC X(16) VALUE 'WRITE           '.    SAM00310
       01 SOKET-FUNCT            PIC X(16) VALUE '                '.    SAM00320
       01 SOKET-ADDR.                                                    SAM00330
         02 SOCK-FAMILY         PIC 9(4) BINARY.                        SAM00340
         02 SOCK-PORT           PIC 9(4) BINARY.                        SAM00350
         02 SOCK-IPADDR         PIC 9(8) BINARY.                        SAM00360
         02 SOCK-ZERO           PIC X(8).                               SAM00370
       01 SOKET-ID               PIC 9(4) BINARY.                        SAM00380
       01 SOKET-ID-NEW           PIC 9(4) BINARY.                        SAM00390
       01 MAXSOC                 PIC 9(4) BINARY.                        SAM00400
       01 IDENT.                                                         SAM00410
          02   TCPNAME           PIC X(8).                               SAM00420
          02   ADSNAME           PIC X(8).                               SAM00430
       01 SUBTASK                PIC X(8).                               SAM00440
       01 MAXSNO                 PIC 9(8) BINARY.                        SAM00450
```

```
                                                                SAM00460
 01 INBUFFL               PIC 9(8) COMP VALUE 40.               SAM00570
                                                                SAM00580
 01 AF-INET               PIC 9(8) COMP VALUE 2.                SAM00470
 01 SOCTYPE               PIC 9(8) COMP VALUE 1.                SAM00480
 01 PROTO                 PIC 9(8) COMP VALUE 0.                SAM00490
 01 BACKLOG               PIC 9(8) COMP VALUE 5.                SAM00500
 01 RETCODE               PIC S9(8) BINARY.                     SAM00510
 01 ERRNO                 PIC 9(8) BINARY.                      SAM00520
 01 MSG001                PIC X(34)                             SAM00530
        VALUE IS '  ... SAMPSERV received your data.'.          SAM00540
 01 MSG001L               PIC 9(8) COMP VALUE 34.               SAM00550
 01 INBUFF                PIC X(40) VALUE  IS ' '.              SAM00560
 PROCEDURE DIVISION.                                            SAM00590
                                                                SAM00600
 BEGIN.                                                         SAM00610
                                                                SAM00620
*-----------------------------------------------*               SAM00630
*     CALL EZASOKET - function = INITAPI       *               SAM00640
*                     input    = SUBTASK blank *               SAM00650
*-----------------------------------------------*               SAM00660
                                                                SAM00670
     MOVE SOKET-INITAPI TO SOKET-FUNCT.                         SAM00680
     MOVE '        '    TO TCPNAME.                            SAM00690
     MOVE '        '    TO SUBTASK.                            SAM00700
     MOVE 99 TO MAXSOC.                                         SAM00710
     MOVE 0 TO RETCODE.                                         SAM00720
     MOVE 0 TO ERRNO.                                           SAM00730
                                                                SAM00740
     CALL 'EZASOKET' USING SOKET-FUNCT MAXSOC IDENT SUBTASK     SAM00750
                 MAXSNO ERRNO RETCODE.                          SAM00760
                                                                SAM00770
*-----------------------------------------*                     SAM00780
*     CALL EZASOKET - function = SOCKET     *                   SAM00790
*-----------------------------------------*                     SAM00800
                                                                SAM00810
     MOVE SOKET-SOCKET TO SOKET-FUNCT.                          SAM00820
     MOVE 0 TO RETCODE.                                         SAM00830
     MOVE 0 TO ERRNO.                                           SAM00840
                                                                SAM00850
     CALL 'EZASOKET' USING SOKET-FUNCT AF-INET SOCTYPE PROTO    SAM00860
                 ERRNO RETCODE.                                 SAM00870
                                                                SAM00880
     MOVE RETCODE TO SOKET-ID.                                  SAM00890
                                                                SAM00900
*-------------------------------------------------- *          SAM00910
*     CALL EZASOKET - function = BIND               *          SAM00920
*                     input    = Soket-id, Soket-addr *        SAM00930
*-------------------------------------------------*            SAM00940
                                                                SAM00950
     MOVE SOKET-BIND TO SOKET-FUNCT.                            SAM00960
     MOVE AF-INET TO SOCK-FAMILY.                               SAM00970
     MOVE 4000    TO SOCK-PORT.                                 SAM00980
     MOVE 0       TO SOCK-IPADDR.                               SAM00990
     MOVE 0 TO RETCODE.                                         SAM01000
     MOVE 0 TO ERRNO.                                           SAM01010
                                                                SAM01020
     CALL 'EZASOKET' USING SOKET-FUNCT  SOKET-ID SOKET-ADDR     SAM01030
                 ERRNO RETCODE.                                 SAM01040
                                                                SAM01050
                                                                SAM01060
*-------------------------------------------------*            SAM01070
*     CALL EZASOKET - function = LISTEN          *            SAM01080
*                     input    = backlog=5       *            SAM01090
*-------------------------------------------------*            SAM01100
                                                                SAM01110
     MOVE SOKET-LISTEN TO SOKET-FUNCT.                          SAM01120
```

```
           MOVE 0 TO RETCODE.                                          SAM01130
           MOVE 0 TO ERRNO.                                            SAM01140
                                                                       SAM01150
           CALL 'EZASOKET' USING SOKET-FUNCT SOKET-ID BACKLOG          SAM01160
                        ERRNO RETCODE.                                 SAM01170
                                                                       SAM01180
                                                                       SAM01190
       *-----------------------------------------*                     SAM01200
       *     CALL EZASOKET - function = ACCEPT     *                   SAM01210
       *                     input    = SOKET-ID   *                   SAM01220
       *-----------------------------------------*                     SAM01230
                                                                       SAM01240
           MOVE SOKET-ACCEPT TO SOKET-FUNCT.                           SAM01250
           MOVE 0 TO RETCODE.                                          SAM01260
           MOVE 0 TO ERRNO.                                            SAM01270
                                                                       SAM01280
           CALL 'EZASOKET' USING SOKET-FUNCT  SOKET-ID SOKET-ADDR      SAM01290
                        ERRNO RETCODE.                                 SAM01300
                                                                       SAM01310
           MOVE RETCODE TO SOKET-ID-NEW.                               SAM01320
                                                                       SAM01330
       *-----------------------------------------------------*         SAM01340
       *     CALL EZASOKET - function = READ                 *         SAM01350
       *-----------------------------------------------------*         SAM01360
                                                                       SAM01370
           MOVE SOKET-READ        TO SOKET-FUNCT.                     SAM01380
           MOVE 0 TO RETCODE.                                          SAM01390
           MOVE 0 TO ERRNO.                                            SAM01400
                                                                       SAM01410
           MOVE LOW-VALUES TO INBUFF.                                  SAM01420
           CALL 'EZASOKET' USING SOKET-FUNCT SOKET-ID-NEW INBUFFL      SAM01430
                 INBUFF ERRNO RETCODE.                                 SAM01440
                                                                       SAM01450
       *-----------------------------------------------------*         SAM01460
       *     CALL EZASOKET - function = WRITE                *         SAM01470
       *-----------------------------------------------------*         SAM01480
                                                                       SAM01490
           MOVE SOKET-WRITE       TO SOKET-FUNCT.                     SAM01500
           MOVE 0 TO RETCODE.                                          SAM01510
           MOVE 0 TO ERRNO.                                            SAM01520
                                                                       SAM01530
           CALL 'EZASOKET' USING SOKET-FUNCT SOKET-ID-NEW MSG001L      SAM01540
                 MSG001 ERRNO RETCODE.                                 SAM01550
                                                                       SAM01560
       *-----------------------------------------------------*         SAM01570
       *     CALL EZASOKET - function = CLOSE                *         SAM01580
       *-----------------------------------------------------*         SAM01590
                                                                       SAM01600
           MOVE SOKET-CLOSE       TO SOKET-FUNCT.                     SAM01610
           MOVE 0 TO RETCODE.                                          SAM01620
           MOVE 0 TO ERRNO.                                            SAM01630
                                                                       SAM01640
           CALL 'EZASOKET' USING SOKET-FUNCT SOKET-ID-NEW              SAM01650
                        ERRNO RETCODE.                                 SAM01660
                                                                       SAM01670
       *-----------------------------------------------------*         SAM01680
       *     CALL EZASOKET - function = CLOSE                *         SAM01690
       *-----------------------------------------------------*         SAM01700
                                                                       SAM01710
           MOVE SOKET-CLOSE       TO SOKET-FUNCT.                     SAM01720
           MOVE 0 TO RETCODE.                                          SAM01730
           MOVE 0 TO ERRNO.                                            SAM01740
                                                                       SAM01750
           CALL 'EZASOKET' USING SOKET-FUNCT SOKET-ID                  SAM01760
                        ERRNO RETCODE.                                 SAM01770
                                                                       SAM01780
                                                                       SAM01790
```

```
*-------------------------------------------------------*          SAM01800
*     CALL EZASOKET - function = TERMAPI                *          SAM01810
*-------------------------------------------------------*          SAM01820
                                                                   SAM01830
     MOVE SOKET-TERMAPI     TO SOKET-FUNCT.                        SAM01840
                                                                   SAM01850
     CALL 'EZASOKET' USING SOKET-FUNCT.                            SAM01860
                                                                   SAM01870
                                                                   SAM01880
     STOP RUN.                                                     SAM01890
                                                                   SAM01900
 END PROGRAM SAMPSERV.                                             SAM01910
```

LE/VSE 1.4 C ソケット・プログラミング

C プログラミングの一般的考慮事項

Language Environment は、OS/390® と z/OS、VM/ESA と z/VM が Language Environment をベースにしたそれぞれの C ランタイム・ライブラリーにおいてカバーしているのと同じ機能をカバーすることを意図していますが、C ソケット・インターフェース・ルーチンの実際の動作は、このインターフェースで使用される TCP/IP 製品に依存しています。従って、別の IBM Z オペレーティング・システム環境からアプリケーションを移植するプログラマーは、VSE ソケット・インターフェースには特別な注意を要するということに後で気付くことがあります。しかし、例えば z/OS からアプリケーションを移植するプログラマーは、ソース・コードの変更が必要になることを予期していない場合があります。

以下のリストは、特にアプリケーションを移植するときには特別な注意が必要な、プログラミング領域を示します。

- C ソケット・インターフェースを使用するアプリケーションを、CICS 環境用に支障なく作成できるようになりました。これは、LE ソケット・サポートが、動的に実行環境を判別し、CICS サービスを状況に応じて適切に使用するからです。例えば、**VSE WAIT** マクロの代わりに **EXEC CICS WAIT** を使用するなどがこれに該当します。このことは、z/OS CICS ソケットとは対照的に、CICS 環境で実行するために、特別な初期化および終了のサービスを C プログラム内で呼び出す必要がないことを暗黙に意味しています。従って、バッチ・アプリケーションまたは CICS アプリケーションのどちらかから呼び出される通信ルーチンを作成することが可能です。
- LE/VSE 1.4 は、複数のサブタスクが LE 対応コードを実行すると想定される場合は、マルチタスク環境をサポートしません。これは z/VSE が、POSIX スレッドも、VSE パーティション当たり 31 を超えるサブタスクもサポートしないためです。また、必要な UNIX 類似システム・インターフェースがないため、新規プロセスを fork() することもできません。それでも、複数の VSE サブタスクを実行させることは可能ですが、LE 対応コードを実行できるのは、そのうちの 1 つだけです。

  従って、複数のクライアントに同時にサービスすることを意図してデーモン・アプリケーションをコーディングする場合、データを読み取ろうとしている間やクライアントが接続するのを待機中に、待機に束縛されないようにすることが必要です。そのため、select() または selectex() を使用して、どのソケットにアクティビティーがあるのかを、recv() または accept() を呼び出す前にチェックする

ことをお勧めします。なぜなら、これらの呼び出しは、その呼び出しの時点である特定のソケット接続上でサービスを受けられる未解決タスクがない場合はブロックする可能性があるためです。

- 他の TCP/IP インプリメンテーションでは、ブロッキング・モードまたは非ブロッキング・モードでソケット・インターフェースが作動することを可能にする、ioctl() または fcntl() インターフェースが提供されています。ブロッキング・モードでは、(例えば recv() への) 呼び出しは、使用されるソケットのデータが到着するまでタスクを中断します。非ブロッキング・モードでは、ルーチンは -1 を戻し、*errno* 変数が **EWOULDBLOCK** に設定されます。そうすると、アプリケーションは、何か別の処理を行うことか、または、select() または selectex() を使用して 1 つまたは複数のソケットがアクティビティーを示すのを待機することのいずれかを選択できます。

  TCP/IP for z/VSE はこのような仕組みをネイティブでは提供していませんが、TCP/IP の LE C ソケット API サポートが、必要なサポートを提供します。ただし、以下の制限が適用されます。

  – 完全に BSD に準拠するスタック・インプリメンテーションでは、デフォルトの送信バッファーおよび受信バッファーが TCP プロトコル用に割り振られています。ネットワークを介してデータを転送できるスタックよりも速く送信バッファーがいっぱいになった場合、send() または sendto() 呼び出しはブロックします。非ブロッキング・モードでは、それらの呼び出しは代わりにエラー値 EWOULDBLOCK を戻します。書き込みビット・ストリングが設定された select() または selectex() 呼び出しは、使用可能なバッファー・スペースがある場合はすぐに戻ります。TCP/IP for z/VSE は、そのようには動作しませんが、パーティション GETVIS を使い尽くすまで、未送信データを GETVIS のバッファーに入れます。未送信データをバッファーするための GETVIS スペースが残っていない場合、send() または sendto() 呼び出しはブロックします。書き込みビット・ストリングが設定された select() または selectex() の呼び出しは、送信バッファー・スペースが使用可能かどうかを示しませんが、すべてのソケット固有の未送信データがネットワークに置かれるまでブロックします。

- 一部の LE/VSE C ソケット・ルーチンには特別な注意が必要です。なぜなら、TCP/IP インプリメンテーションが z/VSE では他のプラットフォームの場合とは異なる動作をするか、または、機能のサブセットのみがインプリメントされているからです。これらの相違点については、 101 ページの『第 11 章 TCP/IP による LE/VSE C ソケット・インターフェースのサポート』の個別の関数の説明で、『スタックの特性』として説明しています。

メッセージ

以下のリストは、LE C ソケット・インターフェース・ルーチンによって発行される可能性のあるメッセージをカバーしています。これらのメッセージは、C ランタイム・ライブラリーによって発行されるか、LE C ソケット呼び出しを TCP/IP for z/VSE BSD-C ソケット・インターフェース・ルーチンにマッピングするときには、フェーズ $EDCTCPV によって発行されます。

LE/VSE 1.4 C ランタイム・ライブラリーによって発行されるメッセージ

- EDCT001I フェーズ $EDCTCPV をロードできません

  フェーズ $EDCTCPV をロードできませんでした。アプリケーションは、メッセージ CEE3322C で取り消されます。

  おそらく、アプリケーションのパーティション LIBDEF 検索チェーンに TCP/IP 製品ライブラリー (PRD2.TCPIPC) が欠落しています。
- EDCT002I xxxxxxxxx インプリメンテーションが見つかりません

  フェーズ $EDCTCPV は、ビルドのエラーが原因で、TCP/IP 機能 xxxxxxxxx の本体を含んでいません。アプリケーションは、CEE3322C で取り消されます。
- EDCT003I サポートされない C ランタイム関数が呼び出されました

  アプリケーションには、LE/VSE 1.4 でサポートでサポートされていない C ランタイム関数の呼び出しが含まれています。アプリケーションは、CEE3322C で取り消されます。

  これが発生するのは、z/OS または z/VM 上でコンパイルおよびプリリンクされたプログラムが、z/VSE 上でリンク・エディットされた場合だけのはずです。z/OS または z/VM® 上のプリコンパイルのステップで、LE/VSE 1.4 ランタイム環境でサポートされていない C ランタイム関数へのスタブ・ルーチンが組み込まれました。

フェーズ $EDCTCPV によって発行されるメッセージ

- EDCV001I TCP/IP 関数 xxxxxxxxx はインプリメントされていません

  アプリケーションは、TCP/IP プログラミング・インターフェースによってインプリメントされていないソケット・ルーチンを呼び出しました。アプリケーションには、関数固有の適当な戻りコードが戻されます。プログラム実行は継続します。

  関数は現在 TCP/IP for z/VSE によってサポートされていないか、または、パーティション LIBDEF チェーンが LE 製品ライブラリーの前に TCP/IP 製品ライブラリーをリストしていません。TCP/IP for z/VSE 製品ライブラリー (PRD2.TCPIPC) からのフェーズ $EDCTCPV は、LE/VSE 製品ライブラリー (PRD2.SCEEBASE) からの同じフェーズよりも前に検出されなければなりません。
- EDCV002I 予期しない TCP/IP エラー・コード: nnnn

  TCP/IP 製品が予期しないエラー・コードを戻しました。LE C インターフェースに対する TCP/IP サポートは処理できません。代わりに、呼び出し側アプリケーションにエラー値 EOPNOTSUPP が戻されます。

z/VSE によってサポートされる TCP/IP 関数

VSE Language Environment 1.4 が提供する C ソケット・インターフェースは、Language Environment (LE) 自体には実装されていませんが、TCP/IP スタックに付属しているプログラミング・インターフェースにマップされています。

表 3 は、 101 ページの『第 11 章 TCP/IP による LE/VSE C ソケット・インターフェースのサポート』で説明している LE C ソケット・ルーチンを示すとともに、それらが以下の TCP/IP スタックのいずれかを通じて現在使用可能であるかどうかを示しています。

- TCP/IP for z/VSE
- IPv6/VSE
- Linux ファスト・パス

この表は、ある関数が一般的にサポート対象であるかどうかを示しているにすぎません。相違点および特殊な特性の詳細は、個々のスタックおよび関数の説明を参照してください。

表 3 は、z/VSE でサポートされる、対応する EZASMI マクロおよび EZASOKET 呼び出しも示しています。これらのインターフェースは、多少の違いはありますが、z/OS でも使用可能です。詳しくは、以下を参照してください。

- 229 ページの『第 12 章 CALL 命令アプリケーション・プログラム・インターフェース (EZASOKET API) の使用』
- 333 ページの『第 13 章 マクロ・アプリケーション・プログラミング・インターフェース (EZASMI API) の使用』

表 *3.* サポートされる呼び出し関数 *(*インターフェースおよび *TCP/IP* スタック別*)*

	インターフェース			**TCP/IP** スタック		
呼び出し関数	**EZASMI**	**EZASOCKET**	**LE/VSE**	**TCP/IP for z/VSE**	**IPv6/ VSE**	**Linux** ファスト・パス
accept()	ACCEPT	ACCEPT	はい	はい	はい	はい
aio_cancel()	CANCEL	いいえ	はい	はい	はい	はい
aio_error()	いいえ	いいえ	はい	いいえ	はい	はい
aio_read()	いいえ	いいえ	はい	いいえ	はい	はい
aio_return()	いいえ	いいえ	はい	いいえ	はい	はい
aio_suspend()	いいえ	いいえ	はい	いいえ	はい	はい
aio_write()	いいえ	いいえ	はい	いいえ	はい	はい
bind()	BIND	BIND	はい	はい	はい	はい
close()	CLOSE	CLOSE	はい	はい	はい	はい
connect()	CONNECT	CONNECT	はい	はい	はい	はい
endhostent()	いいえ	いいえ	はい	いいえ	いいえ	はい
endnetent()	いいえ	いいえ	はい	いいえ	いいえ	はい
endprotoent()	いいえ	いいえ	はい	いいえ	いいえ	はい
endservent()	いいえ	いいえ	はい	いいえ	いいえ	はい
fcntl()	FCNTL	FCNTL	はい	はい	はい	はい
freeaddrinfo()	FREEADDRINFO	FREEADDRINFO	はい	いいえ	はい	はい
gai_strerror()	いいえ	いいえ	はい	いいえ	はい	はい
getaddrinfo()	GETADDRINFO	GETADDRINFO	はい	いいえ	はい	はい
getclientid()	GETCLIENTID	GETCLIENTID	はい	はい	はい	はい
gethostbyaddr()	GETHOSTBYADDR	GETHOSTBYADDR	はい	はい	はい	はい

表 3. サポートされる呼び出し関数 (インターフェースおよび *TCP/IP* スタック別) (続き)

	インターフェース			**TCP/IP** スタック		
呼び出し関数	**EZASMI**	**EZASOCKET**	**LE/VSE**	**TCP/IP for z/VSE**	**IPv6/ VSE**	**Linux** ファスト・パス
gethostbyname()	GETHOSTBYNAME	GETHOSTBYNAME	はい	はい	はい	はい
gethostent()	いいえ	いいえ	はい	いいえ	いいえ	はい
gethostid()	GETHOSTID	GETHOSTID	はい	はい	はい	はい
gethostname()	GETHOSTNAME	GETHOSTNAME	はい	はい	はい	はい
getibmopt()	GETIBMOPT	GETIBMOPT	はい	いいえ	はい	はい
getnameinfo()	GETNAMEINFO	GETNAMEINFO	はい	いいえ	はい	はい
getnetbyaddr()	いいえ	いいえ	はい	いいえ	いいえ	はい
getnetbyname()	いいえ	いいえ	はい	いいえ	いいえ	はい
getnetent()	いいえ	いいえ	はい	いいえ	いいえ	はい
getpeername()	GETPEERNAME	GETPEERNAME	はい	はい	はい	はい
getprotobyname()	いいえ	いいえ	はい	いいえ	いいえ	はい
getprotobynumber()	いいえ	いいえ	はい	いいえ	いいえ	はい
getprotoent()	いいえ	いいえ	はい	いいえ	いいえ	はい
getservbyname()	いいえ	いいえ	はい	いいえ	はい	はい
getservbyport()	いいえ	いいえ	はい	いいえ	はい	はい
getservent()	いいえ	いいえ	はい	いいえ	いいえ	はい
getsockname()	GETSOCKNAME	GETSOCKNAME	はい	はい	はい	はい
getsockopt()	GETSOCKOPT	GETSOCKOPT	はい	はい	はい	はい
givesocket()	GIVESOCKET	GIVESOCKET	はい	はい	はい	はい
gsk_free_memory()	GSKFREEMEM	GSKFREEMEM	はい	はい	いいえ	はい
gsk_get_cipher_info()	GSKGETCIPHINF	GSKGETCIPHINF	はい	はい	いいえ	はい
gsk_get_dn_by_label()	GSKGETDNBYLAB	GSKGETDNBYLAB	はい	はい	いいえ	はい
gsk_initialize()	GSKINIT	GSKINIT	はい	はい	いいえ	はい
gsk_secure_soc_close()	GSKSSOCCLOSE	GSKSSOCCLOSE	はい	はい	いいえ	はい
gsk_secure_soc_init()	GSKSSOCINIT	GSKSSOCINIT	はい	はい	いいえ	はい
gsk_secure_soc_read()	GSKSSOCREAD	GSKSSOCREAD	はい	はい	いいえ	はい
gsk_secure_soc_reset()	GSKSSOCRESET	GSKSSOCRESET	はい	はい	いいえ	はい
gsk_secure_soc_write()	GSKSSOCWRITE	GSKSSOCWRITE	はい	はい	いいえ	はい
gsk_uninitialize()	GSKUNINIT	GSKUNINIT	はい	はい	いいえ	はい
gsk_user_set()	いいえ	いいえ	はい	いいえ	いいえ	はい
htonl()	いいえ	いいえ	はい	はい	はい	はい
htons()	いいえ	いいえ	はい	はい	はい	はい
if_freenameindex()	いいえ	いいえ	はい	いいえ	いいえ	はい
if_indextoname()	いいえ	いいえ	はい	いいえ	いいえ	はい
if_nameindex()	いいえ	いいえ	はい	いいえ	いいえ	はい
if_nametoindex()	いいえ	いいえ	はい	いいえ	いいえ	はい
inet_addr()	いいえ	いいえ	はい	はい	はい	はい

表 3. サポートされる呼び出し関数 (インターフェースおよび *TCP/IP* スタック別) (続き)

	インターフェース			**TCP/IP** スタック		
呼び出し関数	**EZASMI**	**EZASOCKET**	**LE/VSE**	**TCP/IP for z/VSE**	**IPv6/ VSE**	**Linux** ファスト・パス
inet_lnaof()	いいえ	いいえ	はい	はい	いいえ	はい
inet_makeaddr()	いいえ	いいえ	はい	はい	いいえ	はい
inet_netof()	いいえ	いいえ	はい	はい	いいえ	はい
inet_network()	いいえ	いいえ	はい	はい	いいえ	はい
inet_ntoa()	いいえ	いいえ	はい	はい	はい	はい
inet_ntop()	NTOP	NTOP	はい	いいえ	はい	はい
inet_pton()	PTON	PTON	はい	いいえ	はい	はい
initapi()	INITAPI	INITAPI	はい	はい	はい	はい
ioctl()	IOCTL	IOCTL	はい	はい	はい	はい
listen()	LISTEN	LISTEN	はい	はい	はい	はい
maxdesc()	いいえ	いいえ	はい	いいえ	はい	はい
ntohl()	いいえ	いいえ	はい	はい	はい	はい
ntohs()	いいえ	いいえ	はい	はい	はい	はい
poll()	いいえ	いいえ	はい	いいえ	いいえ	はい
read()	READ	READ	はい	はい	はい	はい
readv()	READV	READV	はい	はい	はい	はい
recv()	RECV	RECV	はい	はい	はい	はい
recvfrom()	RECVFROM	RECVFROM	はい	はい	はい	はい
recvmsg()	いいえ	いいえ	はい	いいえ	いいえ	はい
select()	SELECT	SELECT	はい	はい	はい	はい
selectex()	SELECTEX	SELECTEX	はい	はい	はい	はい
send()	SEND	SEND	はい	はい	はい	はい
sendmsg()	いいえ	いいえ	はい	いいえ	いいえ	はい
sendto()	SENDTO	SENDTO	はい	はい	はい	はい
sethostent()	いいえ	いいえ	はい	いいえ	いいえ	はい
setibmopt()	いいえ	いいえ	はい	いいえ	はい	はい
setnetent()	いいえ	いいえ	はい	いいえ	いいえ	はい
setprotoent()	いいえ	いいえ	はい	いいえ	いいえ	はい
setservent()	いいえ	いいえ	はい	いいえ	いいえ	はい
setsockopt()	SETSOCKOPT	SETSOCKOPT	はい	はい	はい	はい
shutdown()	SHUTDOWN	SHUTDOWN	はい	はい	はい	はい
socket()	SOCKET	SOCKET	はい	はい	はい	はい
socketpair()	いいえ	いいえ	はい	いいえ	いいえ	はい
takesocket()	TAKESOCKET	TAKESOCKET	はい	はい	はい	はい
	TASK	いいえ	いいえ	はい	はい	はい
termapi()	TERMAPI	TERMAPI	はい	はい	はい	はい
write()	WRITE	WRITE	はい	はい	はい	はい

表 3. サポートされる呼び出し関数 (インターフェースおよび *TCP/IP* スタック別) (続き)

	インターフェース			**TCP/IP** スタック		
呼び出し関数	**EZASMI**	**EZASOCKET**	**LE/VSE**	**TCP/IP for z/VSE**	**IPv6/ VSE**	**Linux** ファスト・パス
writev()	WRITEV	WRITEV	はい	はい	はい	はい

ERRNO 値

このセクションでは、TCP/IP LE/C、EZASMI/EZASOKET ソケット・インターフェース、またはその両方から返されるすべての ERRNO 値の概要を示します。

- 表 4 に、10 進値でソートした順に ERRNO 値を示します。
- 88 ページの表 5 は、EZASMI/EZASOKET ソケット・インターフェースにのみ適用される ERRNO 値を、10 進値でソートした順に示しています。
- 88 ページの表 6 は、値を ERRNO 名でソートして示しています。

表 4. 値でソートした *ERRNO* 値

ERRNO	**LE/C** または **EZASMI/ EZASOKET** からの **ERRNO** 値	説明
EDOM	1	ドメイン・エラー。
ERANGE	2	範囲エラー。
ELOAD	83	ロード・エラー。
EACCES	111	許可が拒否されました。
EAGAIN	112	リソースは一時的に使用できません。
EBADF	113	正しくないソケット記述子。
EBUSY	114	リソースが使用中。
ECHILD	115	子プロセスなし。
EDEADLK	116	リソース・デッドロックが回避された。
EEXIST	117	ファイルが存在している。
EFAULT	118	正しくないアドレス、またはバッファー・アドレスがアクセス不能。
EFBIG	119	ファイルが大きすぎる。
EINTR	120	関数呼び出しが割り込まれた。
EINVAL	121	無効なパラメーター
EIO	122	ソケットはクローズされている。
EISDIR	123	ディレクトリーである。
EMFILE	124	オープン・ファイルが多すぎる。
EMLINK	125	リンクが多すぎる。
ENAMETOOLONG	126	ファイル名が長すぎる。
ENFILE	127	オープンしているソケットが多すぎる。
ENODEV	128	該当装置はない。
ENOENT	129	そのようなソケットはない。

表 4. 値でソートした *ERRNO* 値 (続き)

ERRNO	**LE/C** または **EZASMI/ EZASOKET** からの **ERRNO** 値	説明
ENOEXEC	130	EXEC 形式エラー。
ENOLCK	131	使用可能なロックがない。
ENOMEM	132	要求を満足するのに十分なメモリーがない。
ENOSPC	133	装置にスペースが残されていない。
ENOSYS	134	関数はインプリメントされていない。
ENOTDIR	135	ディレクトリーではない。
ENOTEMPTY	136	ディレクトリーが空ではない。
ENOTTY	137	不適切な入出力制御操作。
ENXIO	138	該当する装置またはアドレスがない。
EPERM	139	許可されていない命令。
EPIPE	140	パイプが壊れている。
EROFS	141	読み取り専用ファイル・システム。
ESPIPE	142	シークが無効。
ESRCH	143	該当プロセスがない。
EXDEV	144	別のファイル・システム上のファイルへのリンクが試みられた。
E2BIG	145	引数リストが長すぎる。
ELOOP	146	ループが、パス引数の解析中に検出されるシンボリック・リンクに存在する。
EILSEQ	147	無許可のバイト・シーケンス。
ENODATA	148	使用可能なメッセージがない。
EOVERFLOW	149	値が大きすぎてデータ型に保管できない。
EMVSNOTUP	150	OpenEdition がアクティブではない。
EMVSDYNALC	151	動的割り振りエラー。
EMVSCVAF	152	カタログ・ボリューム・アクセス機能エラー。
EMVSCATLG	153	カタログ取得エラー。
EMVSINITIAL	156	プロセス初期化エラー。
EMVSERR	157	内部エラーが発生しました。
EMVSPARM	158	正しくないパラメーター。
EMVSPFSFILE	159	永続ファイル・エラー。
EMVSBADCHAR	160	環境変数名における無効な文字。
EMVSPFSPERM	162	システム・エラー。
EMVSSAFEXTRERR	163	SAF/RACF 抽出エラー。
EMVSSAF2ERR	164	SAF/RACF エラー。

表 4. 値でソートした *ERRNO* 値 (続き)

ERRNO	**LE/C** または **EZASMI/ EZASOKET** からの **ERRNO** 値	説明
EMVSTODNOTSET	165	システム TOD クロックが設定されていない。
EMVSPATHOPTS	166	アクセス・モード引数が PATHOPTS パラメーターと矛盾している。
EMVSNORTL	167	OpenEdition バージョンの C RTL へのアクセスが拒否された。
EMVSEXPIRE	168	パスワードの有効期限が切れた。
EMVSPASSWORD	169	パスワードが無効。
EVSE	183	VSE ではサポートされない。
ELENOFORK	200	Language Environment のメンバーである言語が fork() を許容できない。
ELEMSGERR	201	メッセージ・ファイルが階層ファイル・システムで検出されなかった。
EIBMBADCALL	1000	IUCV ヘッダーにおける無効なソケット呼び出し定数。
EIBMBADPARM	1001	ほかの IUCV ヘッダー・エラー。
EIBMSOCKOUTOFRANGE	1002	割り当てられたソケット番号が範囲外。
EIBMSOCKINUSE	1003	割り当てられたソケット番号が既に使用中。
EIBMIUCVERR	1004	IUCV エラーにより要求が失敗。
EOFFLOADboxERROR	1005	オフロード・ボックスのエラー。
EOFFLOADboxRESTART	1006	オフロード・ボックスが再始動した。
EOFFLOADboxDOWN	1007	オフロード・ボックスがダウン。
EIBMCONFLICT	1008	ソケット上の未解決の呼び出しの競合。
EIBMCANCELLED	1009	要求が取り消された。
ENOTBLK	1100	必要なブロック装置。
ETXTBSY	1101	テキスト・ファイルが使用中。
EWOULDBLOCK	1102	要求はブロックされる。非ブロッキングとマークされたソケット上の操作で、他の場合は機能が実行を中断する結果になるような、使用可能データなしといった状況が発生した。
EINPROGRESS	1103	ソケット接続が進行中。O_NONBLOCK がソケット記述子に設定され、接続を即時に確立できない。
EALREADY	1104	接続要求は既に進行中。接続要求は、指定されたソケットについて既に進行中である。
ENOTSOCK	1105	記述子はソケットを参照しない。

表 4. 値でソートした *ERRNO* 値 (続き)

ERRNO	**LE/C** または **EZASMI/ EZASOKET** からの **ERRNO** 値	説明
EDESTADDRREQ	1106	宛先アドレスが必要。バインド・アドレスが指定されなかった。
EMSGSIZE	1107	メッセージが長すぎる。
EPROTOTYPE	1108	ソケット・タイプがプロトコルでサポートされません。
ENOPROTOOPT	1109	認識できるオプションがない。 setsockopt() に指定されたオプションはサポートされない。
EPROTONOSUPPORT	1110	プロトコルは、アドレス・ファミリーによってサポートされないか、またはインプリメンテーションによってサポートされない。
ESOCKTNOSUPPORT	1111	ソケット・タイプがサポートされていない。
EOPNOTSUPP	1112	ソケット呼び出しがサポートされていない。
EPFNOSUPPORT	1113	プロトコル・ファミリーがサポートされていない。
EAFNOSUPPORT	1114	アドレス・ファミリーはサポートされない (AF_INET 以外)。指定されたアドレス・ファミリーをインプリメンテーションがサポートしないか、または、指定されたアドレスは、指定されたソケットのアドレス・ファミリーに有効なアドレスではない。
EADDRINUSE	1115	指定されたアドレスまたはポートは既に使用中。
EADDRNOTAVAIL	1116	アドレスが利用できない。
ENETDOWN	1117	宛先に到達するため、または使用するためのローカル・インターフェースがダウンしている。
ENETUNREACH	1118	ネットワークに接続できない。
ENETRESET	1119	リセット時にネットワークの接続が切れた。
ECONNABORTED	1120	接続が打ち切られた。
ECONNRESET	1121	接続はピアによって強制的にクローズ/リセットされた。
ENOBUFS	1122	使用可能なバッファーがない。ソケット操作を実行するための十分なバッファー・リソースがシステムで使用可能でない。

表 4. 値でソートした *ERRNO* 値 (続き)

ERRNO	**LE/C** または **EZASMI/ EZASOKET** からの **ERRNO** 値	説明
EISCONN	1123	指定されたソケットは既に接続されている。
ENOTCONN	1124	ソケットは接続されていない。
ESHUTDOWN	1125	ソケット・シャットダウンの後で送信できない。
ETOMANYREFS	1126	参照が多すぎる。継ぎ合わせることができない。
ETIMEDOUT	1127	接続要求がタイムアウトになった。リモート・マシンへの接続はタイムアウトになった。このエラーを報告した関数を実行中に接続がタイムアウトになった場合 (関数が呼び出される前にタイムアウトになるのではなく)、関数が正常に完了した場合の動作の一部またはすべてが完了したかどうかは指定されていない。
ECONNREFUSED	1128	接続が拒否された。
EHOSTDOWN	1129	ホストがダウン。
EHOSTUNREACH	1130	宛先ホストに到達できない。
EPROCLIM	1131	プロセスが多すぎる。
EUSERS	1132	ユーザーが多すぎる。
EDQUOT	1133	予約済み。
ESTALE	1134	ファイル・ハンドルの有効期限が切れた。
EREMOTE	1135	パスのリモート・レベルが多すぎる。
ENOSTR	1136	ストリームではない。
ETIME	1137	ストリーム ioctl() タイムアウト。
ENOSR	1138	ストリーム・リソースがない。
ENOMSG	1139	要求されるタイプのメッセージがない。
EBADMSG	1140	不正なメッセージ。
EIDRM	1141	ID が除去された。
ENONET	1142	マシンがネットワーク上にない。
ERREMOTE	1143	オブジェクトがリモートである。
ENOLINK	1144	リンクが切断された。
EADV	1145	アドバタイズ・エラー。
ESRMNT	1146	srmount エラー。
ECOMM	1147	送信における通信エラー。
EPROTO	1148	プロトコル・エラー。
EMULTIHOP	1149	マルチ・ホップが許可されていない。
EDOTDOT	1150	クロス・マウント・ポイント (エラーではない)。

表 4. 値でソートした *ERRNO* 値 (続き)

ERRNO	**LE/C** または **EZASMI/ EZASOKET** からの **ERRNO** 値	説明
EREMCHG	1151	リモート・アドレスが変更された。
ECANCELED	1152	非同期入出力要求は取り消された。

表 5. *EZASMI/EZASOKET* のみ - 値でソートした *ERRNO* 値

ERRNO	**EZASMI/ EZASOKET** からの **ERRNO** 値	説明	参照する注の番号
EZAINVFU	20000	EZASOKET 呼び出しで無効な関数が使用された。	1
EZAINVPA	20001	EZASOKET 呼び出しで正しくないパラメーター。	1
EZAERL00	20100	フェーズ EZASOH00 のロードでエラー。	
EZAERGTV	20107	不十分なパーティション GETVIS。	
EZAERNIN	20108	最初の呼び出しが INITAPI でない。	
EZAERREC	20111	EZA インターフェースの再帰的エントリー。	
EZAETRNA	20112	EZATRUE がアクティブでない (CICS のみ)。	
EZAERLIF	20113	TCP/IP インターフェース・ルーチンのロードが失敗した。	
EZAERBRI	20114	TCP/IP I/F ルーチンからの戻りコードが正しくない。	

注**:**

1. EZASOKET インターフェースのみが使用します。

プログラミングに関する注意**:**

1. ERRNO (EZASMI/EZASOKET が戻すもの以外) に対する C 言語定義は、PRD2.SCEEBASE に入って出荷される ERRNO.H にあります。
2. EZASMI マクロまたは EZASOKET 呼び出しインターフェースから戻される ERRNO に対するアセンブラーの等価指定は、EZASMI TYPE=TASK,STORAGE=DSECT によってアセンブラー・プログラムに組み込むことができます。

表 6. 名前でソートした *ERRNO* 値

ERRNO	値
EACCESS	111
EADDRINUSE	1115

表 6. 名前でソートした *ERRNO* 値 (続き)

ERRNO	値
EADDRNOTAVAIL	1116
EAFNOSUPPORT	1114
EAGAIN	112
EALREADY	1104
EBADF	113
EBADMSG	1140
EBUSY	114
ECANCELED	1152
ECHILD	115
ECOMM	1147
ECONNABORTED	1120
ECONNREFUSED	1128
ECONNRESET	1121
EDEADLK	116
EDESTADDRREQ	1106
EDOM	1
EDOTEDOT	1150
EDQUOT	1133
EEXIST	117
EFAULT	118
EFBIG	119
EHOSTDOWN	1129
EHOSTUNREACH	1130
EIBMBADCALL	1000
EIBMBADPARM	1001
EIBMCANCELLED	1009
EIBMCONFLICT	1008
EIBMIUCVERR	1004
EIBMSOCKINUSE	1003
EIBMSOCKOUTOFRANGE	1002
EIDRM	1141
EILSEQ	147
EINPROGRESSS	1103
EINTR	120
EINVAL	121
EIO	122
EISCONN	1123
EISDIR	123
ELEMSGERR	201
ELENOFORK	200

表 6. 名前でソートした *ERRNO* 値 (続き)

ERRNO	値
ELOAD	83
ELOOP	146
EMFILE	124
EMLINK	125
EMSGSIZE	1107
EMULTIHOP	1149
EMVSBADCHAR	160
EMVSCATLG	153
EMVSCVAF	152
EMVSSDYNALC	151
EMVSERR	157
EMVSEXPIRE	168
EMVSINITIAL	156
EMVSNORTL	167
EMVSNOTUP	150
EMVSPARM	158
EMVSPASSWORD	169
EMVSPATHOPTS	166
EMVSPFSFILE	159
EMVSPFSPERM	162
EMVSSAF2ERR	164
EMVSSAFEXTRERR	163
EMVSTODNOTSET	165
ENAMETOOLONG	126
ENETDOWN	1117
ENETRESET	1119
ENETUNREACH	1118
ENFILE	127
ENOBUFS	1122
ENODATA	148
ENODEV	128
ENOENT	129
ENOEXEC	130
ENOLINK	1144
ENOLCK	131
ENOMEM	132
ENOMSG	1139
ENONET	1142
ENOPROTOOPT	1109
ENOSPC	133

表 6. 名前でソートした *ERRNO* 値 (続き)

ERRNO	値
ENOSR	1138
ENOSTR	1136
ENOSYS	134
ENOTBLK	1100
ENOTCONN	1124
ENOTDIR	135
ENOTEMPTY	136
ENOTSOCK	1105
ENOTTY	137
ENXIO	138
EOFFLOADboxDOWN	1007
EOFFLOADboxERROR	1005
EOFFLOADboxRESTART	1006
EOPNOTSUPP	1112
EOVERFLOW	149
EPERM	139
EPFNOSUPPORT	1113
EPIPE	140
EPROCLIM	1131
EPROTO	1148
EPROTONOSUPPORT	1110
EPROTOTYPE	1108
ERANGE	2
EREMCHG	1151
EREMOTE	1135
EROFS	141
ERREMOTE	1143
ESHUTDOWN	1125
ESOCKTNOSUPPORT	1111
ESPIPE	142
ESRCH	143
ESRMNT	1146
ESTALE	1134
ETIME	1137
ETIMEDOUT	1127
ETOOMANYREFS	1126
ETXTBSY	1101
EUSERS	1132
EVSE	183
EWOULDBLOCK	1102

表 6. 名前でソートした *ERRNO* 値 (続き)

ERRNO	値
EXDEV	144
E2BIG	145
EZAERBRI	20114
EZAERGTV	20107
EZAERL00	20100
EZAERLIF	20113
EZAERNIN	20108
EZAERREC	20111
EZAETRNA	20112
EZAINVFU	20000
EZAINVPA	20001

CICS の考慮事項

C ソケット・プログラミング・インターフェースは、CICS 実行環境またはバッチ実行環境のいずれかのアプリケーションの作成をサポートします。これは、EZASMI マクロおよび EZASOKET 呼び出しインターフェースの場合も同じです。

ただし、アセンブラー SOCKET マクロおよび TCP/IP for z/VSE HLL プリプロセッサー (**EXEC TCP** 呼び出しを解決する) が明示的に実行環境を指定するのに対して、BSD-C ソケット・インターフェースではそれは可能ではありません。

プログラマーは、CICS プログラムまたはバッチ・プログラムから呼び出される、2 モード対応のモジュールまたはアプリケーションを作成できます。TCP/IP ランタイム・サービスは、実行環境の要件に合わせて動作します。つまり、適切であれば、最終的には CICS サービス (例えば、**EXEC CICS WAIT**) を使用します。

アプリケーションがその実行環境を動的に判定するようにするには、アプリケーションのリンク・エディットのステップに次の 2 つの OBJ ファイルを組み込む必要があります。

- IPCICSRQ (TCP/IP for z/VSE のみ)
- DFHECI

これらの 2 つのファイルを省略すると、アプリケーションは、CICS の制御下で実行している場合でも、CICS に適さない動作をする結果になります。これは例えば、CICS GETMAIN の代わりに VSE GETVIS 要求を発行する場合などが該当します。

注**:** これは、TCP/IP for z/VSE の BSD-C インターフェースを使用する非 LE ソケット・アプリケーションにも当てはまります。VSE Language Environment 1.4 C ランタイムで提供される C ソケット・インターフェースは、上記の目的でこれらのモジュールをリンクする必要はありません。これは既に、アプリケーションに透過的に、TCP/IP for z/VSE による LE/VSE 1.4 C ソケット・インターフェー

スのサポートによってカバーされています。このサポートについては、 101 ページの『第 11 章 TCP/IP による LE/VSE C ソケット・インターフェースのサポート』を参照してください。

EZA インターフェースに関する CICS の考慮事項

CICS TS トランザクション環境で EZA API (EZASMI マクロおよび EZASOKET 呼び出しインターフェース) が使用できるのは、その「タスク関連ユーザー出口」(TRUE) ルーチンが開始された後です。この EZA 「タスク関連ユーザー出口」の名前は、EZATRUE です。このルーチンは、タスク関連の作業用ストレージを EZA API 処理環境に割り振り、CICS タスク終了処理中にクリーンアップを行います。

「タスク関連ユーザー出口」EZATRUE は、以下のいずれかの方法で、プログラム EZASTRUE と共に開始/終了されます。

- トランザクション EZAT (EZAT START が EZATRUE を開始し、EZAT STOP が停止します)
- PLTPI (CICS スタートアップ中の自動スタートアップ) および PLTSD (CICS シャットダウン中の自動シャットダウン) への EZASTRUE のエントリー
- 以下の COMMAREA パラメーターを指定した、プログラム EZASTRUE への EXEC CICS LINK

表 *7. COMMAREA* パラメーター・リスト

オフセット	長さ	説明	
0	8	目印「EZATRUE」	
8	1	要求タイプ	
		「S」	開始要求
		「T」	終了要求
9	1	EZASTRUE からの戻りコード	
		0	EZATRUE 開始/終了は成功した
		4	EZATRUE は既に要求された状態である
		8	EZATRUE の開始/終了は失敗した
		16	無効なパラメーター・リスト

TCP/IP アプリケーション・プログラムの実行

TCP/IP への接続

デフォルトで、TCP/IP アプリケーションは、ID=00 が割り当てられた TCP/IP パーティションとの接続を試みます。TCP/IP スタックの ID の割り当て方法については、対応する TCP/IP スタックの資料を参照してください。デフォルトの ID 値は「00」です。「00」以外の ID を使用して TCP/IP パーティションに接続する必要がある場合は、JCL に適切な OPTION ステートメントを組み込むことで接続できるようになります。

```
// OPTION SYSPARM='xx'
```

ここで、xx は 2 桁の ID 番号であり、TCP/IP スタートアップ JCL またはパラメーターと同じになるようにコーディングします。

SSL/TLS のための準備とセットアップ

セキュア・ソケット通信のための LE/VSE C、EZASMI および EZASOKET 関数呼び出しを使用する前に、VSE システムで SSL for VSE が使用できるように準備する必要があります。

注: SSL for VSE は、TCP/IP for z/VSE を使用する場合のみ使用可能です。

この準備作業には、以下のものがあります。

- (オプション) 秘密鍵および証明書が保管されるライブラリーおよびサブライブラリーの作成 (ディスク上のデフォルト・ファイルを使用しない場合)
- (オプション) 秘密鍵および証明書用に使用されるライブラリー、サブライブラリー、およびメンバー名の定義 (ディスク上のデフォルト・ファイルを使用しない場合)
- 秘密鍵の作成
- サーバー証明書の作成
- ルート証明書の作成
- (オプション) SSL for VSE 証明書の検査

デフォルトの SSL/TLS セットアップについては「*IBM z/VSE e-business Connectors* ユーザーズ・ガイド」を、この準備作業の詳細については「*TCP/IP for VSE Optional Features*」を参照してください。

第 10 章 使用する TCP/IP と SSL の実装を選択

z/VSE は、TCP/IP 区画 (スタック) への接続時にソケット・アプリケーションに柔軟性を提供します。 TCP/IP スタックの他、使用する SSL 実装も選択できます。

TCP/IP への接続

デフォルトで、TCP/IP アプリケーションは、ID=00 が割り当てられた TCP/IP パーティションとの接続を試みます。TCP/IP スタックの ID の割り当て方法については、対応する TCP/IP スタックの資料を参照してください。デフォルトの ID 値は「00」です。「00」以外の ID を使用して TCP/IP パーティションに接続する必要がある場合は、JCL に適切な OPTION ステートメントを組み込むことで接続できるようになります。

```
// OPTION SYSPARM='xx'
```

ここで、xx は 2 桁の ID 番号であり、TCP/IP スタートアップ JCL またはパラメーターと同じになるようにコーディングします。

また、ご使用のアプリケーションの LIBDEF に、TCP/IP スタックがインストールされているライブラリーが含まれていることを確認する必要があります。

- **PRD2.TCPIPC** (TCP/IP for z/VSE の場合)。
- **PRD2.TCPIPB** (IPv6/VSE の場合)。
- **IJSYSRS.SYSLIB** Linux Fast Path (LIBDEF 内で自動的に)。

LE/C ソケット API を使用した TCP/IP への接続

LE/VSE C ランタイム・ソケット API (LE/C ソケット API) を使用する場合は、// OPTION SYSPARM で指定されている ID を、C ランタイム環境変数 SYSID=xx で上書きできます。

環境変数は、ランタイム・オプション ENVAR または関数 setenv() を使用して設定することができます。ランタイム・オプション ENVAR について詳しくは、「*LE/VSE Programming Reference*」および「*LE/VSE Customization Guide*」を参照してください。関数 setenv() について詳しくは、「*LE/VSE C Run-Time Library Reference*」を参照してください。

EZA API を使用した TCP/IP への接続

EZASOKET / EZASMI INITAPI 関数呼び出しを使用する場合は、IDENT.TCPNAME 入力パラメーターを使用して、接続先の TCP/IP 区画を選択できます。この 8 バイトのパラメーターは、「SOCKET*nn*」または単に「*nn*」と設定できます (左寄せまたは右寄せし、6 つのブランクを埋め込む)。値「*nn*」は、選択された TCP/IP スタックの ID を決定します。これにより、// OPTION SYSPARM の指定がオーバーライドされます。

EZASOKET および EZASMI のインターフェースを使用すると、TCP/IP 区画に対して使用するソケット・インターフェース・モジュールを指定できます。デフォルトとして、EZASOH99 (TCP/IP for z/VSE のソケット・インターフェース・モジュール) が使用されます。

以下の 2 つの方法で、異なる EZA ソケット・インターフェース・ルーチンの使用を要求できます。

- JCL ステートメント `// SETPARM [SYSTEM,]EZA$PHA='phasename'` を使用して。
- EZAAPI/EZASOKET INITAPI 呼び出しでパラメーター ADSNAME を使用して。

指定の順序で検査が行われます。 JCL パラメーター EZA$PHA も INITAPI パラメーター ADSNAME も設定しない場合は、デフォルトの EZA ソケット・インターフェース・ルーチン EZASOH99 が使用されます。

以下の EZA ソケット・インターフェース・ルーチンが使用できます。

- **EZASOH99** (TCP/IP for z/VSE の場合)。
- **BSTTIPS1** (IPv6/VSE 用)。
- **IJBLFPEZ** (Linux Fast Path 用)。
- **IJBEZAMX** (EZA マルチプレクサー使用時)。

ソケット API マルチプレクサーの使用

LE/C ソケット API または EZA API を使用する場合は、構成済みスタック ID で正しいソケット・インターフェース・モジュールが使用されていなければなりません。例えば TCP/IP for z/VSE スタックで IPv6/VSE ソケット・インターフェース・モジュールを使用しようとしている場合、これは失敗します。

対応するスタック ID に対するソケット・インターフェース・モジュールの正しいセットアップを容易に行うために、ソケット API マルチプレクサーを使用できます。マルチプレクサーを使用すると、一回限りのセットアップを実行し、対応するソケット・インターフェース・モジュールをスタック ID に割り当てることができます。マルチプレクサーの使用は、ご使用のアプリケーションにとってトランスペアレントです。

また、マルチプレクサーを使用して、ソケット・アプリケーションでどの SSL 実装が使用されるかを構成することもできます。デフォルトでは、割り当てられたソケット・インターフェース・モジュールによって提供される SSL 実装が使用されます。マルチプレクサーを使用すると、スタック ID ごとに代替 SSL 実装を指定できます。ただし、代替 SSL 実装の使用はご使用のアプリケーションにとってトランスペアレントであり、鍵および証明書のセットアップは SSL 実装ごとに異なります。

マルチプレクサー構成は LE/C ソケット API にも EZA ソケット API にも使用されます。

マルチプレクサー構成は一般的に PRD2.CONFIG に保管されます。このファイルはすべてのソケット・アプリケーションの LIBDEF チェーン内にあることが必要で

す。構成はテーブルをアセンブルすることによって生成されます。マルチプレクサー構成の例については、ICCF ライブラリー 62 にあるスケルトン EDCTCPMC を参照してください。

マルチプレクサー構成項目は EDCTCPME マクロを使用して作成されます。マクロでは以下のパラメーターがサポートされます。

SYSID (必須)
2 文字の数値スタック ID を「00」から「99」の範囲で指定します。

PHASE (必須)
LE/C ソケット・インターフェース・モジュールの名前を指定します。使用可能な LE/C ソケット・インターフェース・モジュールのリストについては、98 ページの『LE/C ソケット API によるマルチプレクサーの使用』を参照してください。

SSLPHASE (オプション)
LE/C SSL 実装の名前を指定します。指定しない場合は、PHASE パラメーターで指定された LE/C ソケット・インターフェース・モジュールの名前にデフォルト設定されます。使用可能な LE/C SSL 実装のリストについては、98 ページの『LE/C ソケット API によるマルチプレクサーの使用』を参照してください。

EZAPHASE (オプション)
EZA ソケット・インターフェース・モジュールの名前を指定します。指定しない場合は、EZASOH99 にデフォルト設定されます。使用可能な EZA ソケット・インターフェース・モジュールのリストについては、99 ページの『EZA ソケット API によるマルチプレクサーの使用』を参照してください。

EZASSL (オプション)
EZA SSL 実装の名前を指定します。指定しない場合は、EZAPHASE パラメーターで指定された EZA ソケット・インターフェース・モジュールの名前にデフォルト設定され、EZAPHASE も指定されていない場合は EZASOH99 にデフォルト設定されます。使用可能な EZA SSL 実装のリストについては、99 ページの『EZA ソケット API によるマルチプレクサーの使用』を参照してください。

マルチプレクサー構成で指定されていないスタック ID のデフォルトは以下のようになります。

- LE/C ソケット・インターフェースと LE/C SSL の場合: **$EDCTCPV**。
- EZA ソケット・インターフェースと EZA SSL の場合: **EZASOH99**。

使用可能なマルチプレクサー構成がまったくない場合は、同じデフォルトが適用されます。

同じ ID に複数の項目が指定されている場合は、最初の項目のみが使用され、同じ ID の以降の項目は無視されます。

マルチプレクサー構成の例:

```
* $$ JOB JNM=EDCTCPMC,CLASS=A,DISP=D
// JOB EDCTCPMC GENERATE TCP/IP MULTIPLEXER CONFIG PHASE
// LIBDEF *,CATALOG=PRD2.CONFIG
```

```
// LIBDEF *,SEARCH=(PRD1.BASE)
// OPTION ERRS,SXREF,SYM,NODECK,CATAL,LISTX
  PHASE EDCTCPMC,*,SVA
// EXEC ASMA90,SIZE=(ASMA90,64K),PARM='EXIT(LIBEXIT(EDECKXIT)),SIZE(MAXC
               -200K,ABOVE)'
EDCTCPMC CSECT
EDCTCPMC AMODE ANY
EDCTCPMC RMODE ANY
*
         EDCTCPME SYSID='00',PHASE='$EDCTCPV',SSLPHASE='IJBSSLLE',     X
               EZAPHASE='EZASOH99',EZASSL='IJBEZAOS'
         EDCTCPME SYSID='01',PHASE='IJBLFPLE',SSLPHASE='IJBSSLLE',     X
               EZAPHASE='IJBLFPEZ',EZASSL='IJBEZAOS'
         EDCTCPME SYSID='02',PHASE='BSTTTCPV',SSLPHASE='IJBSSLLE',     X
               EZAPHASE='BSTTIPS1',EZASSL='IJBEZAOS'
         EDCTCPME SYSID='03',PHASE='VENDTCPI',SSLPHASE='IJBSSLLE',     X
               EZAPHASE='VENDEZAI',EZASSL='IJBEZAOS'
*
         END
/*
// IF $MRC GT 4 THEN
// GOTO NOLINK
// EXEC LNKEDT,PARM='MSHP'
/. NOLINK
/*
/&
* $$ EOJ
```

LE/C ソケット API によるマルチプレクサーの使用

LE/C ソケット API の場合、マルチプレクサーは常にアクティブです。マルチプレクサーを活動化するために、これ以上の手順は不要です。マルチプレクサー構成が使用できない場合は、デフォルト (上記参照) が適用されます。

以下の LE/C ソケット・インターフェース・モジュールが使用できます。

- **$EDCTCPV** (TCP/IP for z/VSE 用、デフォルト)。
- **BSTTTCP6** (IPv6/VSE 用)。
- **IJBLFPLE** (Linux Fast Path 用)。

3 以下の代替 SSL/TLS 実装が使用できます。

- **IJBSSLLE** (OpenSSL 用)。 OpenSSL サポートについて詳しくは、 487 ページの『第 17 章 OpenSSL』を参照してください。

C ランタイム環境変数 _EDCTCPM_DEBUG が定義されている場合、マルチプレクサーは、アクティブ定義を検査するために役立つ可能性があるデバッグ・メッセージを標準出力 (CICS の SYSLST または CESO) に発行します。

TCP/IP for z/VSE を使用したソケット API のデバッグの場合はさらに、$EDCTCPV の代わりに $DBGTCPV を使用することができます。これにより、より詳細なデバッグ・メッセージが標準エラー出力 (CICS の SYSLST または CESE) に生成されます。

EZA ソケット API によるマルチプレクサーの使用

EZA マルチプレクサーは、デフォルトではアクティブになりません。 EZA マルチプレクサーを活動化するには、システム・パラメーター `// SETPARM [SYSTEM,]EZA$PHA='IJBEZAMX'` を設定する必要があります。システムで実行されているすべてのアプリケーションに適用されるように、システム・レベルでこのパラメーターを設定することをお勧めします。あるいは、EZASMI / EZASOKET INITAPI 呼び出しで IDENT.ADSNAME='IJBEZAMX' を設定して、アプリケーション用に明示的に EZA マルチプレクサーを活動化することもできます。

以下の EZA ソケット・インターフェース・ルーチンが使用できます。

- **EZASOH99** (TCP/IP for z/VSE 用、デフォルト)。
- **BSTTIPS1** (IPv6/VSE 用)。
- **IJBLFPEZ** (Linux Fast Path 用)。

以下の代替 SSL 実装が使用できます。

- **IJBEZAOS** (OpenSSL 用)。 OpenSSL サポートについて詳しくは、 487 ページの『第 17 章 OpenSSL』を参照してください。

第 11 章 TCP/IP による LE/VSE C ソケット・インターフェースのサポート

概要

BSD ソケット (バークレー・ソケット) は、UNIX プラットフォーム向けに開発された、TCP/IP プログラミング・インターフェースを使用する方法の 1 つです。特に UNIX 類似プラットフォーム用には他のソケット・ルーチンもありますが、それらのルーチンのサブセットのみがインプリメントされます。TCP/IP for z/VSE が提供する、BSD-C に類似したインターフェースは、主に、非 LE 対応 C コンパイラー (例えば、IBM C/370 コンパイラー) のユーザー向けに用意されたものです。このインターフェースの説明は、「*TCP/IP for VSE Programmer's Guide*」に記載されています。

IBM C for VSE/ESA リリース 1 (5686-A01) コンパイラーを IBM Language Environment for z/VSE (LE/VSE) 1.4 C ランタイム環境と一緒に使用する場合、LE/VSE 1.4 ソケット・インターフェースの使用を強くお勧めします。これらのインターフェースは、OS/390 X/Open (XPG4.2) 準拠のソケット・インターフェースと互換性があります。このことにより、プラットフォーム間の開発において、最高の互換性と移植性が確保されます。

注:

1. LE/VSE 1.4 ランタイム環境は、ソケット・ルーチン自体をインプリメントしませんが、TCP/IP for z/VSE 製品の一部で PRD2.TCPIPC に保管されているフェーズ $EDCTCPV を動的に呼び出します。従って、ソケット・アプリケーションは、TCP/IP 製品から分離されます (詳しくは、 59 ページの図 12 を参照してください)。LE/VSE 1.4 ランタイムは、このフェーズを呼び出すことによって新しいサービス・レベルを動的に選出します。一方、ネイティブ TCP/IP BSD- C ソケット・ルーチンは、TCP/IP サービスが適用されると再リンクが必要になります。
2. LE/VSE 1.4 C Base は、TCP/IP for z/VSE がインストールされていないか、削除されたシステム向けに、デフォルトの $EDCTCPV フェーズを PRD2.SCEEBASE に収容して出荷します。このデフォルトのフェーズは、関数固有の戻りコードの定義と、メッセージ EDCV001I (呼び出された関数がインプリメントされていないことを示す) の発行以外には、何も実行しません。

   このメッセージを受け取った場合、アプリケーションの LIBDEF が正しいかどうか検査し、ルーチンが使用可能であると想定されるかどうかチェックしてください。
3. LE/VSE 1.4 C ランタイムは OS/390 と同じ範囲のソケット・ルーチン群を提供していますが、TCP/IP for z/VSE がインプリメントするのはそのサブセットのみです。このことは、LE/VSE C ランタイム・インターフェースを使用する場合は、その使用法とインプリメンテーション詳細について本章を参照する必要があることを意味します。

4. LE/C ソケット API マルチプレクサーを使用して、アプリケーションで使用する TCP/IP および SSL の実装を選択する方法については、 95 ページの『第 10 章 使用する TCP/IP と SSL の実装を選択』を参照してください。

TCP/IP 呼び出し可能関数 — 関数の説明

accept() — ソケットでの新規接続の受け入れ

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

int accept(int socket, struct sockaddr *address, size_t *address_len);
```

概要

accept() 呼び出しは、クライアントからの接続要求を受け入れるために、サーバーによって使用されます。詳しくは、ご使用の TCP/IP プロバイダーの機能説明を参照してください。接続が使用可能になると、作成されたソケットは、接続要求を行ったプロセスからデータを読み取る準備ができます。呼び出しは、指定されたソケット *socket* に対する保留接続の待ち行列にある最初の接続を受け入れます。
accept() 呼び出しは *socket* と同じ特性を持つ新しいソケット記述子を作成して、それを呼び出し元に戻します。元のソケットである *socket* は、今までどおり接続要求を受け入れることができます。

パラメーター
: 説明

socket
: ソケット記述子。

address
: 接続するクライアントのソケット・アドレスであり、戻る前に accept() によって完成します。 *address* の形式は、クライアントが存在しているドメインによって決まります。呼び出し元がクライアント・アドレスを必要としない場合、このパラメーターを NULL にすることができます。

address_len
: 最初は *address* が指すストレージのサイズ (バイト) を含む整数を示していなければなりません。戻るときには、その整数に、*address* が指すストレージに戻されたデータのサイズが入ります。*address* が NULL であると、*address* は、無視されます。

socket パラメーターは、socket() 呼び出しで作成されるストリーム・ソケット記述子です。これは、bind() 呼び出しを使ってアドレスにバインドされます。listen() 呼び出しは、接続を受け入れるソケットであるとソケットをマークし、保留接続要求を保持するための待ち行列を割り振ります。listen() 呼び出しは、待ち行列のサイズの上限を設定します。

address パラメーターは、接続リクエスターのアドレスが置かれるバッファーを指すポインターです。*address* パラメーターはオプションで、NULL ポインターになる

ように設定することができます。NULL に設定すると、リクエスターのアドレスはバッファーにコピーされません。*address* の正確な形式は、通信要求の発信元のアドレッシング・ドメインに依存します。

例えば、接続要求の発信元が AF_INET ドメインの場合、*address* は、sockaddr_in 構造体を指し、接続要求の発信元が AF_INET6 ドメインの場合は、*address* は、sockaddr_in6 構造体を指します。 sockaddr_in および sockaddr_in6 構造体は、**in.h.** 内で定義されています。*address_len* パラメーターは、*name* が NULL でない場合にのみ使用されます。`accept()` を呼び出す前に、*address_len* で指される整数を、*address* で指されるバッファー・サイズ (バイト) に設定しなければなりません。正常終了して戻るときには、*address_len* で指される整数に、バッファーにコピーされた実際のバイト数が入っています。バッファーが、アドレスを保持するのに十分な大きさでない場合、リクエスターのアドレスの *address_len* バイトまでがコピーされます。アドレスの実際の長さが、供給される *sockaddr* の長さを超える場合、保管されるアドレスは切り捨てられます。保管構造体の *sa_len* メンバーには、切り捨てられていないアドレスの長さが含まれます。

注**:** この呼び出しは、SOCK_STREAM ソケットだけで使用されます。`accept()` を呼び出さずにリクエスターを選別する方法はありません。アプリケーションは、どのリクエスターからの接続を受け入れるのかをシステムに通知することはできません。ただし、呼び出し元はリクエスターの ID を検出したら即時に接続をクローズすることを選択できます。

`select()` 呼び出しを使用して、着信接続要求がないかどうかソケットを確認できます。

戻り値

負ではないソケット記述子は成功を表し、値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
: 説明

EBADF
: *socket* パラメーターが、ソケット記述子の許容範囲内にありません。

EFAULT
: 指定された *address* と *address_len* を使用すると、呼び出し元のアドレス・スペースの情報を書き込むことができない部分にアドレスをコピーしようとする結果になります。

EINVAL
: `listen()` が、ソケット記述子 *socket* に対して呼び出されませんでした。

ENFILE
: システムでの最大数のソケット記述子が既にオープンしています。

ENOBUFS
: バッファー・スペースが不足しているため新しいソケットを作成できません。

EOPNOTSUPP
指定されたソケットのソケット・タイプは、接続の受け入れをサポートしません。

EWOULDBLOCK
ソケット記述子 *socket* は非ブロッキング・モードであり、待ち行列に入っている接続はありません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

例

次に、accept() 呼び出しの 2 つの例を示します。最初の例では、呼び出し元はリクエスターのアドレスを戻そうとしています。次の例では、呼び出し元はリクエスターのアドレスを戻そうとしていません。

```
int clientsocket;
int s;
struct sockaddr clientaddress;
int address_len;
int accept(int s, struct sockaddr *addr, int *address_len);
/* socket(), bind(), and listen() have been called */

/* EXAMPLE 1: I want the address now */
address_len = sizeof(clientaddress);
clientsocket = accept(s, &clientaddress, &address_len);

/* EXAMPLE 2: I can get the address later using getpeername() */
clientsocket = accept(s, (struct sockaddr *) 0,
(int *) 0);
```

aio_cancel() — 非同期入出力要求の取り消し

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <aio.h>

int aio_cancel(int socket, struct aiocb *aiocbp);
```

概要

aio_cancel() 関数は、ソケット記述子 socket に対する現在未処理の 1 つ以上の非同期入出力要求を取り消そうとします。*aiocbp* 引数は、取り消される特定の要求の *aiocb* 構造体を指します。または、*socket* に対する取り消し可能な未処理の要求すべてを取り消す場合は NULL です。

正常に取り消された非同期入出力操作については、通常の非同期通知が起こります。関連するエラー状況は ECANCELED に設定され、取り消された要求について戻り状況は -1 に設定されます。

取り消すことのできない要求については、その入出力の完了時に通常の非同期完了処理が行われます。この場合、*aiocb* が aio_cancel() によって変更されることはありません。

非同期操作を取り消すことができるのは、その操作が現在ブロックされているかまたはブロック状態になる場合です。未処理の要求は、いったん実行可能になれば、完了が許可されます。例えば、aio_cancel() が呼び出された時点で使用可能なデータがない場合には、aio_read() を取り消すことができます。

socket は、有効なソケットファイル記述子でなければなりません。しかし、*aiocbp* が NULL でないときは、非同期操作を開始したソケット記述子と *socket* が一致する必要はありません。ただし、移植性を最大にするためには、一致するべきです。

aio_cancel() 関数は常に、取り消されようとしている要求が完了するか、取り消されるまで待機します。aio_cancel() から制御が戻ると、プログラムは、元の要求の aiocb とバッファーを安全に解放できます。

指定した記述子に関して要求をすべて取り消しても、要求を新たに行うなど、その記述子に影響する操作は停止されません。プログラムは、取り消しを発行した理由に応じて、その記述子を再開またはクローズできます。

個々の要求を取り消すことができるのは、1 回だけです。続けて同じ要求を明示的に取り消そうとすると、EALREADY で失敗します。

戻り値

aio_cancel() 関数は、以下のいずれかの値を戻します。

戻り値 説明

AIO_CANCELED
要求した操作は取り消されました。

AIO_NOTCANCELED
要求した操作の少なくとも 1 つが進行中であるために取り消すことができません。この場合、aio_cancel() への呼び出しで参照されるその他の操作 (もし、あれば) の状態は、aio_cancel() の戻り値によっては示されません。aio_error() を使用すると、アプリケーションでその他の操作の状況を判別できます。

AIO_ALLDONE
操作がすでに完了しています。この値が戻されるのは、指定した基準と一致する未処理の要求が見つからなかったときです。また、これは、*socket* に関連付けられたファイルが非同期入出力関数をサポートしないときに戻される結果でもあります。そのような場合には、指定した基準と一致し、検出されるような未処理の要求はないからです。

-1 エラーが発生しました。errno には、エラーのタイプを示す値が設定されます。

aio_cancel() 関数は、以下の場合に失敗します。

errno 説明

EBADF
socket 引数が無効なソケット記述子です。

EALREADY
取り消しをする操作は、すでに取り消されています。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EBADF に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

aio_error() — 非同期入出力操作のエラー状況の取得

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <aio.h>

int aio_error(const struct aiocb *aiocbp);
```

概要

aio_error() 関数は、*aiocb* 引数で参照される *aiocbp* 構造体に関連したエラー状況を戻します。非同期入出力操作のエラー状況は、対応する read() または write() 操作で設定される errno 値と同じです。操作がまだ完了していない場合、エラー状況は EINPROGRESS に等しくなります。

戻り値

非同期入出力操作が正常終了した場合は、0 が戻ります。非同期入出力操作が異常終了した場合は、read() または write() に対する説明と同じエラー状況が戻ります。非同期入出力操作が未完了の場合は、EINPROGRESS が戻ります。

aio_error() 関数では、errno は、設定されません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

aio_read() — ソケットからの非同期読み取り

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <aio.h>

int aio_read(struct aiocb *aiocbp);
```

概要

aio_read() 関数は、*aiocb* 構造体 (非同期入出力制御ブロック) によって記述されている非同期 read 操作を開始します。

aiocbp 引数は、*aiocb* 構造体を指します。この構造体には以下のメンバーが含まれています。

aio_ filedes
: ソケット記述子

aio_offset
: ファイル・オフセット

aio_buf
バッファーの位置

aio_nbytes
転送の長さ

aio_reqprio
要求優先順位オフセット

aio_sigevent
シグナルの番号と値

aio_lio_opcode
実行される操作

この操作では、*aio_ filedes* に関連したソケットから *aio_buf* が指すバッファーへの、*aio_nbytes* までの読み取りが行われます。要求が開始されたかまたは待ち行列に入れられると (データを即時に送達できないときでも)、aio_read() の呼び出しは戻ります。

現在、非同期入出力は、ソケットについてだけサポートされています。*aio_offset* フィールドは、設定しても無視されます。

ストリーム・ソケットでは、データの最初のパケットが着信したときに非同期読み取りが完了する場合があり、アプリケーションは、必要とするデータすべてを取得するために、非同期的または同期的に追加読み取り要求を発行しなければならない場合があります。データグラム・ソケットにはメッセージ境界があり、メッセージ全体が着信するまでは操作は完了しません。

aiocbp 値を aio_error() 関数または aio_return() 関数の引数として使用して、非同期操作のエラー状況または戻り状況を判別することができます。操作の進行中には、aio_error() が取得するエラー状況は EINPROGRESS であり、aio_return() が取得する戻り状況は予測不能です。

キューイングの途中でエラー条件が発生した場合、要求を開始することも待ち行列に入れることもなく、関数呼び出しは戻ります。

プログラムは、結果が EINPROGRESS ではなくなるまで、aio_error() によって *aiocb* を時々ポーリングすることができます。

操作は aio_read() 呼び出しから制御が戻る前に完了する場合があることに注意してください。操作が迅速に完了したときでも、aio_read() の呼び出しからの戻り値は 0 になり、入出力自体の結果ではなく、入出力要求のキューイングが反映されます。

非同期操作は、完了する前に aio_cancel() によって取り消される場合があります。取り消された操作は、ECANCELED というエラー状況で終了します。しかし、タイミング上の理由により、aio_cancel() で取り消される前に、操作が自然に正常終了または異常終了する場合もあります。

この操作のソケット記述子がクローズされると、操作が完了しなかったかまたは完了する直前でない場合には、操作は削除されます。Close() は、その記述子に対応する進行中の非同期操作が削除されるかまたは完了するまで待ちます。

aio_suspend() を使用して、非同期操作が完了するまで待つこともできます。

ソケットは、ブロッキング状態でなければなりません。そうでないと、操作は EWOULDBLOCK で失敗することがあります。

非同期入出力が完了する前に、aiocbp が指す制御ブロックまたは *aio_buf* が指すバッファーが不当なアドレスになると、aio_read() の動作は予測できなくなります。

同じ aiocbp を使用する同時非同期操作、無効な *aiocbp* を使用する非同期操作、またはプロセスのメモリー・スペースのアドレス範囲に対する非同期入出力が未解決である間にそのスペースを変更するシステム処置では、予測できない結果が生じます。

aio_lio_opcode フィールドは、関数 aio_read() によって LIO_READ に設定されます。

_POSIX-PRIORITIZED_IO はサポートされていません。*aio_reqprio* フィールドは、設定しても無視されます。

_POSIX_SYNCHRONIZED_IO はサポートされていません。

戻り値

aio_read() 関数は、入出力操作が正常に待ち行列に入れられた場合は、呼び出し元プロセスにゼロ値を戻し、それ以外の場合は値 - 1 を戻して、エラーを示す値を errno に設定します。aio_read() 関数は、以下の場合に失敗します。

errno　説明

ENOSYS
aio_filedes に関連したファイルが、aio_read() 関数をサポートしていません。

以下の条件のそれぞれは、aio_read() の呼び出しの時点で同期的に検出される場合も、非同期的に検出される場合もあります。以下の条件のいずれかが同期的に検出された場合、aio_read() 関数は -1 を戻し、errno を対応する値に設定します。以下の条件のいずれかが非同期的に検出された場合は、非同期操作の戻り状況は -1 に設定され、非同期操作のエラー状況は、対応する値に設定されます。

エラー状況
説明

EBADF
*aio_ filedes*引数は、読み取り用にオープンされた有効なソケット記述子ではありません。

EWOULDBLOCK
aio_ filedes に関連したファイルは非ブロッキング状態であり、使用可能なデータがありません。

EINVAL
aio_sigevent に無効値が入っています。

aio_read() 関数が入出力操作を正常に待ち行列に入れたが、その操作がその後取り消されたか、またはエラーが検出された場合、非同期操作の戻り状況は、-1 に設定

されます。また、非同期操作のエラー状況は、read() 関数呼び出しで通常設定されるエラー状況に設定されるか、あるいは次の値に設定されます。

エラー状況
: 説明

ECANCELED
: 要求した入出力は、aio_cancel() を明示的に呼び出したために入出力が完了する前に取り消されました。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は ENOSYS に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

aio_return() — 非同期入出力操作の状況の取得

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <aio.h>

int aio_return(const struct aiocb *aiocbp);
```

概要

aio_return() 関数は、*aiocbp* 引数で参照される *aiocb* 構造体に関連した戻り状況を戻します。非同期入出力操作の戻り状況は、対応する read() 操作または write() 操作で設定される値と同じです。操作の進行中には、aio_error() が取得するエラー状況は EINPROGRESS であり、aio_return() が取得する戻り状況は予測不能です。ある非同期操作について、aio_error() が 0 で戻った後に、戻り状況を取得するために aio_return() 関数を呼び出すことができます。

戻り値

非同期入出力操作が正常に完了した場合、read() または write() の場合の説明と同じ戻り状況が戻されます。非同期入出力操作がまだ完了していない場合は、戻り状況は予測できません。

aio_return() では、errno は、設定されません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

aio_suspend() — 非同期入出力要求の待機

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <aio.h>

int aio_suspend(const struct aiocb * const list[ ],
                      int nent, const struct timespec * timeout);
```

概要

aio_suspend() 関数は、*timeout* が NULL ポインターのときは、*list* 引数が指す非同期入出力操作のうち少なくとも 1 つが完了するまで、呼び出し元スレッドを中断します。あるいは、*timeout* が NULL でない場合は、*timeout* で指定した時間間隔が経過するまで中断します。*list* が指す入出力操作のいずれかが完了するよりも前に、*timeout* が指す *timespec* 構造体に指定された時間間隔が経過した場合、aio_suspend() はエラーで戻ります。呼び出しの時点でリスト内の *aoicb* 構造体のいずれかが、完了した非同期入出力操作に対応する (つまり、操作のエラー状況が EINPROGRESS ではない) 場合、この関数は、呼び出し元スレッドを中断することなく戻ります。

list 引数は、非同期入出力制御ブロック (AIOCB) を指すポインターの配列です。*nent* 引数は、配列内のエレメントの数を指示します。ポインターが指す各 *aiocb* 構造体は、非同期入出力要求を開始するときに使用されることになります。この配列に NULL ポインターが含まれることがありますが、それは無視されます。非同期入出力をサブミットするときに使用されなかった *aiocb* 構造体、または無効な *aiocb* 構造体を参照するポインターがこの配列に含まれていると、結果は予測できなくなります。

戻り値

1 つ以上の非同期入出力操作が完了した後に aio_suspend() 関数が戻る場合、この関数はゼロを戻します。それ以外の場合、この関数は値 -1 を戻し、エラーを示す errno を設定します。アプリケーションは、関連するエラー状況と戻り状況を、それぞれ aio_error() または aio_return() を使用してスキャンすることによって、どの非同期入出力が完了したのかを判別できます。具体的なエラーを errno の値が示します。

errno　説明

ENOSYS
: z/VSE が、aio_suspend 関数をサポートしていません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は ENOSYS に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

使用上の注意

1. AIOCB ポインターのリストで表される AIOCB は、aio_suspend の呼び出し元のキーと同じストレージ・キーの中に存在しなければなりません。AIOCB ポインター・リストまたはリストで表される AIOCB のいずれかに呼び出し元がアクセスできないと、EFAULT になる場合があります。
2. 値が 0 のリスト内の AIOCB ポインターは無視されます。
3. タイムアウト値 0 (秒 + ナノ秒) は、aio_suspend() 呼び出しでまったく待機が行われないことを意味します。この呼び出しでは、完了した非同期入出力要求について調べられます。そのような要求が見つからないと、この関数は EAGAIN により戻ります。そのような要求が少なくとも 1 つ見つかると、aio_suspend() は成功して戻ります。

4. tv_sec フィールドが <limits.h> に定義されている INT_MAX に設定された *timespec* のタイムアウト値では、aio_suspend サービスは非同期入出力要求が完了するまで待つ結果となります。
5. aio_suspend() に渡される AIOCB は、当該サービスがまだ進行中であるときに解放されたり再使用されたりしてはなりません。非同期入出力が完了したあとでも、当該サービスが AIOCB を使用することがあります。aio_suspend() の途中で AIOCB を変更すると、予測できない結果が生じます。

aio_write() — ソケットへの非同期書き込み

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <aio.h>

int aio_write(struct aiocb *aiocbp);
```

概要

aio_write() 関数は、*aiocb* 構造体 (非同期入出力制御ブロック) によって記述されている非同期 write 操作を開始します。

aiocbp 引数は、*aiocb* 構造体を指します。この構造体には以下のメンバーが含まれています。

aio_ filedes
: ソケット記述子

aio_offset
: ファイル・オフセット

aio_buf
: バッファーの位置

aio_nbytes
: 転送の長さ

aio_reqprio
: 要求優先順位オフセット

aio_sigevent
: シグナルの番号と値

aio_lio_opcode
: 実行される操作

この操作では、*aio_buf* が指すバッファーから *aio_ filedes* に関連したソケットに、*aio_nbytes* の書き込みが行われます。要求が開始されたかまたは待ち行列に入れられると (データを即時に送達できないときでも)、aio_write() の呼び出しは戻ります。

現在、非同期入出力は、ソケットについてだけサポートされています。*aio_offset* フィールドは、設定しても無視されます。

aiocbp 値を aio_error() 関数または aio_return() 関数の引数として使用して、非同期操作のエラー状況または戻り状況を判別することができます。操作の進行中には、aio_error() が取得するエラー状況は EINPROGRESS であり、aio_return() が取得する戻り状況は予測不能です。

キューイングの途中でエラー条件が発生した場合、要求を開始または待ち行列に入れることなく、関数呼び出しは戻ります。

プログラムは、結果が EINPROGRESS ではなくなるまで、aio_error() によって *aiocb* を時々ポーリングすることができます。

操作は aio_read() 呼び出しから制御が戻る前に完了する場合があることに注意してください。操作が迅速に完了したときでも、aio_read() の呼び出しからの戻り値は 0 になり、入出力自体の結果ではなく、入出力要求のキューイングが反映されます。

非同期操作は、完了する前に aio_cancel() によって取り消される場合があります。取り消された操作は、ECANCELED というエラー状況で終了します。しかし、タイミング上の理由により、aio_cancel() で取り消される前に、操作が自然に正常終了または異常終了する場合もあります。

この操作のソケット記述子がクローズされると、操作が完了しなかったかまたは完了する直前でない場合には、操作は削除されます。Close() は、その記述子に対応する進行中の非同期操作が削除されるかまたは完了するまで待ちます。

aio_suspend() を使用して、非同期操作が完了するまで待つこともできます。ソケットは、ブロッキング状態でなければなりません。そうでないと、操作は EWOULDBLOCK で失敗することがあります。

非同期入出力が完了する前に、*aiocbp* が指す制御ブロックまたは *aio_buf* が指すバッファーが不当なアドレスになると、*aio_read()* の動作は予測できなくなります。

同じ *aiocbp* を使用する同時非同期操作、無効な *aiocbp* を使用する非同期操作の試行、またはプロセスのメモリー・スペースのアドレス範囲に対する非同期入出力が未解決である間にそのスペースを変更するシステム処置では、予測できない結果が生じます。

aio_lio_opcode フィールドは、LIO_WRITE に設定する必要があります。

_POSIX-PRIORITIZED_IO はサポートされていません。*aio_reqprio* フィールドは、設定しても無視されます。

_POSIX_SYNCHRONIZED_IO はサポートされていません。

戻り値

aio_write() 関数は、入出力操作が正常に待ち行列に入れられた場合は、呼び出し元プロセスにゼロ値を戻し、それ以外の場合は値 - 1 を戻して、エラーを示す値を errno に設定します。aio_write() 関数は、以下の場合に失敗します。

errno　説明

ENOSYS
aio_ filedes に関連したファイルは、aio_write() 関数をサポートしません。

以下の条件のそれぞれは、aio_write() の呼び出しの時点で同期的に検出される場合も、非同期的に検出される場合もあります。以下の条件のいずれかが同期的に検出された場合、aio_write() 関数は -1 を戻し、errno を対応する値に設定します。以下の条件のいずれかが非同期的に検出された場合は、非同期操作の戻り状況は -1 に設定され、非同期操作のエラー状況は、対応する値に設定されます。

エラー状況/**errno**
説明

EBADF
aio_ filedes 引数は、書き込み用にオープンされた有効なソケット記述子ではありません。

EWOULDBLOCK
aio_ filedes に関連したファイルは非ブロッキング状態であり、使用可能なデータがありません。

EINVAL
aio_nbytes が無効な値であるか、*aio_sigevent* に無効値が入っています。

aio_write() 関数が入出力操作を正常に待ち行列に入れたが、その操作がその後取り消されたか、またはエラーが検出された場合、非同期操作の戻り状況は、-1 に設定されます。また、非同期操作のエラー状況は、write() 関数呼び出しで通常設定されるエラー状況に設定されるか、あるいは次の値に設定されます。

エラー状況
説明

ECANCELED
要求した入出力は、aio_cancel() を明示的に呼び出したために入出力が完了する前に取り消されました。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は ENOSYS に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

bind() — ソケットへの名前の結合

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

int bind(int socket, const struct sockaddr *name, size_t namelen);
```

概要

bind() 呼び出しは、固有のローカル名を、記述子が *socket* であるソケットに結合します。socket() を呼び出すと、記述子から関連付けられた名前がなくなります。ただし、記述子は、socket() が呼び出されたときに指定された特定のアドレス・ファミリーに所属はしています。名前の正確な形式は、アドレス・ファミリーに依存します。

パラメーター
　　説明

socket　直前の socket() 呼び出しで戻されたソケット記述子。

name　*socket* に結合される名前が入っている **sockaddr** 構造体を指すポインター。

namelen
　　name のサイズ (バイト単位)。

socket パラメーターは、socket() 呼び出しによって作成された、任意のタイプのソケット記述子です。

name パラメーターは、*socket* に結合される名前が入っているバッファーを指すポインターです。*namelen* パラメーターは、*name* が指すバッファーのサイズ (バイト単位) です。

AF_INET ドメインで作成されたソケット記述子

ソケット記述子 *socket* が AF_INET ドメインで作成された場合、名前バッファーの形式は、組み込みファイル **in.h** に定義されているように、**sockaddr_in** であると想定されます。この構造体は、以下のように定義されます。

```
struct in_addr
{
        ip_addr_t s_addr;
};

 struct sockaddr_in {
     unsigned char  sin_len;
     unsigned char  sin_family;
     unsigned short sin_port;
     struct in_addr sin_addr;
     unsigned char  sin_zero[8];

};
```

sin_family フィールドは、AF_INET に設定する必要があります。

sin_port フィールドは、アプリケーションの結合先であるポートに設定します。このフィールドは、ネットワーク・バイト順に指定しなければなりません。*sin_port* を 0 に設定すると、呼び出し元は、使用可能なポートを割り当てるために、このフィールドをシステムに任せます。アプリケーションは、getsockname() を呼び出すことで、割り当てられたポート番号を見つけることができます。

sin_addr.s_addr フィールドは IP アドレスに設定し、ネットワーク・バイト・オーダーで指定する必要があります。複数のネットワーク・インターフェースを持つホスト (マルチホーム・ホストと呼ばれる) では、呼び出し元は、結合先のインターフェースを選択することができます。次に、(結合された名前と一致する) このインターフェースからの UDP パケットおよび TCP 接続要求だけが、アプリケーションに転送されます。このフィールドを、**in.h** で定義されている定数 INADDR_ANY に設定すると、呼び出し元は、ホスト上のすべてのネットワーク・インターフェースにソケットが結合されるよう要求します。続いて、(結合された名前と一致する) すべてのインターフェースからの UDP パケットおよび TCP 接続が、アプリケーションに転送されます。1 つのサーバーが複数のネットワークにサービスを提供するときには、これが重要になります。アドレスを未指定のままにすることにより、

サーバーは、要求の到達先のネットワーク・インターフェースにかかわらず、そのポート用に作成された UDP パケットおよび TCP 接続要求をすべて受け入れることができます。

sin_zero フィールドは使用されません。また、このフィールドは、すべてゼロに設定する必要があります。

AF_INET6 ドメインで作成されたソケット記述子

ソケット記述子 *socket* が AF_INET6 ドメインで作成された場合、名前バッファーの形式は、組み込みファイル **in.h** に定義されているように、**sockaddr_in6** であると想定されます。この構造体は、以下のように定義されます。

```
struct sockaddr_in6 {
   uint8_t sin6_len;
   sa_family_t sin6_family;
   in_port_t sin6_port;
   uint32_t sin6_flowinfo;
   struct in6_addr sin6_addr;
   uint32_t sin6_scope_id;
};
```

sin6_len フィールドは、この構造体のサイズにセットされます。 SIN6_LEN マクロは、使用されている sockaddr_in6 構造体のバージョンを表すように定義されています。

sin6_family フィールドは、これを sockaddr_in6 構造体と識別します。このフィールドは、バッファーが sockaddr 構造体にキャストされた時に *sa_family* フィールドをオーバーレイします。このフィールドの値は、AF_INET6 でなければなりません。

sin6_port フィールドは、16 ビット UDP または TCP ポート番号を含んでいます。このフィールドは、sockaddr_in 構造体の sin_port フィールドと同じ方法で使用されます。ポート番号は、ネットワーク・バイト順に保管されます。

sin6_flowinfo フィールドは、32 ビットのフィールドで、トラフィック・クラスとフロー・ラベルを含んでいます。

sin6_addr フィールドは、単一の in6_addr 構造体です。このフィールドは、1 つの 128 ビット IPv6 アドレスを保持しています。アドレスは、ネットワーク・バイト順に保管されます。

sin6_scope_id フィールドは、32 ビットの整数で、sin6_addr フィールドに繰り上がったアドレスのスコープに見合ったインターフェースのセットを示します。これは、リンク・スコープ sin6_addr、sin6_scope_id のインターフェース・インデックスです。これは、サイト・スコープ sin6_addr、sin6_scope_id のサイト ID です。

戻り値

値 `0` は成功を表します。値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
　　説明

EADDRINUSE
アドレスは既に使用中です。

EAFNOSUPPORT
アドレス・ファミリーがサポートされていません (これは AF_INET または AF_INET6 ではありません)。

EBADF
socket パラメーターが無効ソケット記述子です。

EINVAL
ソケットがすでにアドレスに結合済みです。例えば、すでに接続済みのソケットに名前を結合しようとしています。または、ソケットがシャットダウンされました。

ENOBUFS
bind() は、ストレージが不十分なためにバッファーを取得することができません。

EOPNOTSUPP
指定されたソケットのソケット・タイプがアドレスへの結合をサポートしません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

例

以下の例は、AF_INET ドメインでインターフェースに結合する bind() 呼び出しを示します。IP アドレスとポートは、ネットワーク・バイト・オーダーでなければなりません。ポートをネットワーク・バイト・オーダーにするには、htons() ユーティリティー・ルーチンを呼び出して、ホスト・バイト・オーダーからネットワーク・バイト・オーダーに短整数を変換します。*address* フィールドは、別のユーティリティー・ルーチン inet_addr() を使用して設定します。このルーチンは、インターフェースのドット 10 進アドレスを表す文字ストリングを受け取って、2 進の IP アドレス表記をネットワーク・バイト・オーダーで戻します。要求された名前が予約済みフィールドを何も設定しないようにするため、構造体を使用する前に構造体をゼロにするのは適切な方法です。

```
int rc;
int s;
struct sockaddr_in myname;

/* Bind to a specific interface in the Internet domain */
/* make sure the sin_zero field is cleared */
memset(&myname, 0, sizeof(myname));
myname.sin_family = AF_INET;
myname.sin_addr.s_addr = inet_addr("129.5.24.1");
/* specific interface */
myname.sin_port = htons(1024);
:
rc = bind(s, (struct sockaddr *) &myname,
sizeof(myname));
/* Bind to all network interfaces in the Internet domain */
/* make sure the sin_zero field is cleared */
```

```
memset(&myname, 0, sizeof(myname));
myname.sin_family = AF_INET;
myname.sin_addr.s_addr = INADDR_ANY; /* specific interface */
myname.sin_port = htons(1024);
:
rc = bind(s, (struct sockaddr *) &myname,
sizeof(myname));
aslr.* Bind to a specific interface in the Internet domain.
   Let the system choose a port                          */
/* make sure the sin_zero field is cleared */
memset(&myname, 0, sizeof(myname));
myname.sin_family = AF_INET;
myname.sin_addr.s_addr = inet_addr("129.5.24.1");
/* specific interface */
myname.sin_port = 0;
:
rc = bind(s, (struct sockaddr *) &myname,
sizeof(myname));
```

close() — ソケットのクローズ

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <unistd.h>

int close(int socket);
```

概要

close()呼び出しは、*socket* 記述子に関連したソケットをシャットダウンして、そのソケットに割り振られたリソースを解放します。*socket* がオープン TCP 接続を示す場合、その接続がクローズされます。待ち行列に入っている入力データがあるときに、ストリーム・ソケットがクローズされると、TCP 接続は正しくクローズされるのではなくリセットされます。

パラメーター
　　説明

socket　クローズされるソケットの記述子。

注: プロセスの終了前に、すべてのソケットをクローズしてください。

SO_LINGER ソケット・オプションを使用する AF_INET および AF_INET6 ストリーム・ソケット (SOCK_STREAM) の場合、close が発行されたときにデータがまだ残っていると、ソケットは即時に終了しません。このオプションの設定または設定解除には、以下の構造体が使用されます。これは、**socket.h** に定義され、*setsockopt* ルーチンで使用されます。

```
struct linger {
   int l_onoff;      /* zero=off, nonzero=on */
   int l_linger;     /* time is seconds to linger */
};
```

l_onoff スイッチがゼロ以外の場合、システムは、未送信メッセージを送達しようとします。リンガー時間が指定されている場合、システムは *n* 秒だけ待ってから、データをフラッシュしてソケットを終了します。

戻り値

close() は、成功した場合は 0 を戻します。成功しなかった場合は、-1 を戻し、errno を以下のいずれかに設定します。

EBADF
socket パラメーターが無効ソケット記述子です。

EIO ソケットの読み取り中または書き込み中に、入出力エラーが発生しました。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EBADF に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

connect() — ソケットの接続

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

int connect(int socket, const struct sockaddr *name, size_t namelen);
```

概要

ストリーム・ソケットの場合、connect() 呼び出しは 2 つのソケット間の接続を確立しようとします。データグラム・ソケットの場合、connect() 呼び出しはソケットのピアを指定します。*socket* パラメーターは、接続要求の発信に使用されるソケットです。connect() 呼び出しは、ストリーム・ソケット用に呼び出されたときに、2 つのタスクを実行します。まず、この呼び出しはストリーム・ソケットに必要なバインディングを完了します (bind() 呼び出しを使用して事前にバインディングが完了していない場合)。次に、この呼び出しは別のソケットへの接続を試行します。

パラメーター
説明

socket ソケット記述子。

name 接続の試行先のソケット・アドレスを含むソケット・アドレス構造体を指すポインター。

namelen
name が指すソケット・アドレスのサイズ (バイト単位)。

ストリーム・ソケットでの connect() 呼び出しは、サーバーへの接続を確立するため、クライアント・アプリケーションで使用されます。サーバーには、受動オープン保留が必要です。ソケットを使用しているサーバーは、サーバーが accept() を使用して接続を受け入れられるようになる前に、bind() および listen() を正常に呼び出しておかなければなりません。

socket がブロッキング・モードの場合、connect() 呼び出しは、接続がセットアップされるまで、あるいはエラーが受信されるまで、呼び出し元をブロックします。ソケットが非ブロッキング・モードの場合、connect() は -1 を戻し、エラー・コードを EINPROGRESS に設定して、接続が開始されたがまだ完了していないこと

を示します (エラーが発生しなかった場合)。呼び出し元は、select() を呼び出して、ソケットへの書き込みができるかどうかをテストすることによって、接続セットアップの完了をテストすることができます。

データグラム・ソケット用に呼び出されると、connect() はこのソケットが関連しているピアを指定します。これにより、アプリケーションは接続状態にあるソケット用に確保されたデータ転送呼び出しを使うことができます。この場合、sendto() および recvfrom() 呼び出しに加えて、read()、write()、readv()、writev()、send()、および recv() 呼び出しが使用できるようになります。ストリーム・ソケットは connect() を一度しか呼び出せませんが、データグラム・ソケットはソケットの関連を変更するのに、何度も connect() を呼び出すことができます。データグラム・ソケットは、NULL アドレス (全フィールドがゼロ) などの間違ったアドレスに接続することによって、ソケットの関連を解消することができます。

name パラメーターは、アプリケーションが接続する必要のあるピアの名前を含むバッファーを指すポインターです。*namelen* パラメーターは、*name* が指すバッファーのサイズ (バイト単位) です。

AF_INET ドメイン内のサーバー

サーバーが AF_INET ドメイン内にある場合、名前バッファーの形式は、組み込みファイル **in.h** に定義されているように、**sockaddr_in** であると想定されます。

```
struct in_addr
{
        ip_addr_t s_addr;
};

 struct sockaddr_in {
     unsigned char  sin_len;
     unsigned char  sin_family;
     unsigned short sin_port;
     struct in_addr sin_addr;
     unsigned char  sin_zero[8];

};
```

sin_family フィールドは、AF_INET に設定する必要があります。*sin_port* フィールドは、サーバーの結合先であるポートに設定します。このフィールドは、ネットワーク・バイト順に指定しなければなりません。*sin_zero* フィールドは使用されません。また、このフィールドは、すべてゼロに設定する必要があります。

AF_INET6 ドメイン内のサーバー

サーバーが AF_INET6 ドメイン内にある場合、名前バッファーの形式は、組み込みファイル **in.h** に定義されているように、**sockaddr_in6** であると想定されます。

```
:
struct sockaddr_in6 {
   uint8_t char sin6_len;
   sa_family_t sin6_family;
   in_port_t sin6_port;
   uint32_t sin6_flowinfo;
   struct in6_addr sin6_addr;
   uint32_t sin6_scope_id;
];
```

sin6_ ファミリー は、AF_INET6 にセットされる必要があります。

戻り値

値 0 は成功を表します。値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
: 説明

EAFNOSUPPORT
: アドレス・ファミリーがサポートされません。

EALREADY
: *socket* ソケット記述子が非ブロッキングであるとマークされていて、前の接続の試行が完了していません。

EBADF
: *socket* パラメーターが無効ソケット記述子です。

EFAULT
: *name* および *namelen* を使用すると、呼び出し側のアドレス・スペース内のデータを書き込めない部分へ、アドレスをコピーしようとする結果となります。

EINPROGRESS
: O_NONBLOCK がソケットのソケット記述子に対して設定され、接続を即時に確立できません。その接続は非同期的に確立されます。EINPROGRESS 値はエラー状態を示すものではありません。

EINVAL
: *namelen* パラメーターが有効な長さではありません。

EISCONN
: *socket* ソケット記述子がすでに接続済みです。

EOPNOTSUPP
: *socket* パラメーターが SOCK_STREAM 型ではありません。

ETIMEDOUT
: 接続が行われる前に、接続の確立がタイムアウトとなりました。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

例

以下に示すのは、connect() 呼び出しの例です。IP アドレスとポートは、ネットワーク・バイト・オーダーでなければなりません。ポートをネットワーク・バイト・オーダーにするには、htons() ユーティリティー・ルーチンを呼び出して、ホスト・バイト・オーダーからネットワーク・バイト・オーダーに短整数を変換します。*address* フィールドは、別のユーティリティー・ルーチン inet_addr() を使用して設定します。このルーチンは、インターフェースのドット 10 進アドレスを表す文字ストリングを受け取って、2 進の IP アドレス表記をネットワーク・バイト・オーダーで戻します。最後に、要求された名前が予約済みフィールドを何も設定しないようにするため、構造体を使用する前に構造体をゼロにすることをお勧め

します。以下の例は、 113 ページの『bind() — ソケットへの名前の結合』呼び出しの記載例に示してあるサーバーへの接続に使用できます。

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>
#include <in.h>

int s;
struct sockaddr_in inet_server;
int rc;

/* Connect to server bound to a specific interface in the
Internet domain */
/* make sure the sin_zero field is cleared */
memset(&inet_server, 0, sizeof(inet_server));
inet_server.sin_family = AF_INET;
inet_server.sin_addr = inet_addr("129.5.24.1");
/* specific interface */
inet_server.sin_port = htons(1024);
.
.
.
rc = connect(s, (struct sockaddr *) &inet_server, sizeof(inet_server));
```

endhostent() — ホスト・エントリーの作業

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

void endhostent(void);
```

概要

endhostent() 呼び出しは、既知のホストに関する情報を含むデータ・セットをクローズします。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

endnetent() — ネットワーク情報データ・セットのクローズ

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

void endnetent(void);
```

概要

endnetent() 呼び出しは、既知のネットワークに関する情報を含むデータ・セットをクローズします。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

endprotoent() — プロトコル・エントリーの作業

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

void endprotoent(void);
```

概要

endprotoent() 呼び出しは、ネットワーク・プロトコルに関する情報を含むデータ・セットをクローズします。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

endservent() — ネットワーク・サービス情報データ・セットのクローズ

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

void endservent(void);
```

概要

endservent() 呼び出しは、ネットワーク・サービスに関する情報を含むデータ・セットをクローズします。サービスの例には、ネーム・サーバー、ファイル転送プロトコル (FTP)、Telnet があります。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

fcntl() — オープンされたソケット記述子の制御

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <types.h>
#include <unistd.h>
#include <fcntl.h>

int fcntl(int socket, int cmd, ... /* arg */);
```

概要

fcntl() 呼び出しで、ソケットの操作特性を制御できます。制御される操作は、*cmd* で決定されます。*arg* パラメーターは、*cmd* パラメーターの値によって異なる意味を持つ変数です。

パラメーター
　　説明

socket　ソケット記述子。

cmd　実行するコマンド。

arg　*cmd* に関連するデータ。

cmd 引数は、以下のシンボルのうちの 1 つです。

F_GETFL
このコマンドは、ソケット記述子 *socket* の状況フラグを取得します。_XOPEN_SOURCE_EXTENDED 1 フィーチャー・テスト・マクロで、O_NDELAY フラグを照会できます。O_NDELAY フラグは、*socket* が非ブロッキング・モードにあるとマークします。read()、readv()、recv() などのブロックする可能性のある呼び出しにデータが存在しない場合、呼び出しは -1 を戻し、エラー・コードは EWOULDBLOCK に設定されます。

F_SETFL
このコマンドは、ソケット記述子 *socket* の状況フラグを設定します。_XOPEN_SOURCE_EXTENDED 1 フィーチャー・テスト・マクロで、O_NDELAY フラグを設定できます。

戻り値

成功した場合、戻される値は、指定された *cmd* によって異なります。正常に実行されなかった場合は、fcntl() は -1 を戻し、errno に次のいずれかを設定します。

エラー・コード
説明

EBADF
socket パラメーターが無効ソケット記述子です。

EINVAL
arg パラメーターが有効なフラグでないか、または *cmd* パラメーターが有効なコマンドではありません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

例

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <types.h>
#include <unistd.h>
#include <fcntl.h>
int s;
int rc;
int flags;
:
/* Place the socket into nonblocking mode */
rc = fcntl(s, F_SETFL, O_NDELAY);

/* See if asynchronous notification is set */
flags = fcntl(s, F_GETFL, 0);
if (flags & O_NDELAY)
   /* it is set */
else
   /* it is not */
```

freeaddrinfo() - addrinfo ストレージの解放

フォーマット

```
#define _OPEN_SYS_SOCK_IPV6
#include <socket.h>
#include <netdb.h>


void *freeaddrinfo(struct addrinfo *ai);
```

概要

freeaddrinfo() 関数は、getaddrinfo() によって戻された 1 つ以上の addrinfo 構造体と、それらに関連付けられた追加ストレージを解放します。構造体の ai_next フィールドが非ヌルの場合、リストの全構造体が解放されます。

戻り値

戻り値は定義されません。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

gai_strerror() - アドレスおよび名前情報のエラーの説明

フォーマット

```
#define _OPEN_SYS_SOCK_IPV6
#include <netdb.h>


char *gai_strerror(int ecode);
```

概要

gai_strerror() 関数は、getaddrinfo() または getnameinfo() 関数のどちらかから、障害リターンで戻されたエラー値を記述しているテキスト・ストリングを指すポインターを戻します。*ecode* が <netdb.h> ヘッダーからの EAI_xxx 値の 1 つでない場合は、gai_strerror() は不明なエラーを示すストリングを指すポインターを戻します。

gai_strerror() の以降の呼び出しは、テキスト・ストリングを含むバッファーを上書きします。

戻り値

正常に実行された場合、gai_strerror() はエラーを説明するストリングを指すポインターを戻します。失敗した場合、gai_strerror() は NULL を戻し、errno を以下の 1 つに設定します。

エラー・コード
: 説明

ENOMEM
: エラーを説明するテキスト・ストリングへのバッファーの割り振りに、メモリーが不十分です。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

getaddrinfo() - アドレス情報の取得

フォーマット

```
#define _OPEN_SYS_SOCK_IPV6
#include <socket.h>
#include <netdb.h>


int getaddrinfo(const char *nodename,
                const char *servname,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

概要

getaddrinfo() 関数は、サービス・ロケーション名 (例えば、ホスト名) およびサービス名のいずれか、あるいは双方の変換を行います。ここで戻されるソケット・アドレスのセットと関連情報は、指定されたサービスに対応するためのソケットの作成に使用されます。

nodename および *servname* 引数は、ヌル終了ストリングを指すポインターか、または NULL ポインターです。これらの 2 つの引数の内の 1 つあるいは両方は、NULL ポインター以外に指定されている必要があります。

有効名の形式は、1 つ以上のプロトコル・ファミリーに従属します。特定のファミリーが与えられておらず、その名前が複数のサポートされるファミリーで有効と解釈される場合、その関数は、すべてのサポートされるファミリーの中で名前を解決しようと試みます。エラーがまったく検出されない場合、すべての成功した結果が戻されます。

nodename 引数がヌルでない場合、それは記述名か、またはアドレス・ストリングです。指定されたアドレス・ファミリーが AF_INET、AF_INET6、または AF_UNSPEC の場合、有効な記述名にはホスト名が含まれます。指定されたアドレス・ファミリーが AF_INET または AF_UNSPEC の場合、inet_addr() で指定された標準ドット表記を使用したアドレス・ストリングは有効です。指定されたアドレス・ファミリーが AF_INET6 または AF_UNSPEC の場合、inet_pton() で説明された標準 IPv6 テキスト・フォームは有効です。さらに、スコープ情報は、*nodename%scope* 情報の形式を使用して、記述名またはアドレス・ストリングに追加できます。スコープ情報は、このシステムでの使用に適した、インターフェース・インデックスのインターフェース名または数値表現のいずれかとすることができます。

nodename がヌルでない場合、nodename が要求されたサービス・ロケーションの名前を付けます。そうでない場合は、要求されたサービス・ロケーションは呼び出し元にとってローカルです。

servname がヌルの場合、呼び出しは指定された *nodename* のネットワーク・レベルのアドレスを戻します。*servname* がヌルでない場合、それは要求されたサービスを識別するヌル終了文字ストリングです。これは、1 つ以上のアドレス・ファミリー

と共に使用するのに適切な記述名か、または数値表現です。指定されたアドレス・ファミリーが、AF_INET、AF_INET6、または AF_UNSPEC の場合、サービスは、10 進数ポート番号を指定するストリングとして指定することができます。

引数 *hints* がヌルでない場合、それは入力データ値を含む構造体を参照します。その入力データ値は、オプションを提供すること、および特定のソケット・タイプと、アドレス・ファミリーまたはプロトコル (あるいはその両方) への戻り情報を制限することにより、操作を指示します。hints 構造体では、*ai_flags*、*ai_family*、*ai_socktype*、および *ai_protocol* 以外の各メンバーは、ゼロまたはヌルである必要があります。ai_family の AF_UNSPEC 値は、呼び出し元がすべてのプロトコル・ファミリーを受け入れることを示します。ai_socktype の値がゼロの場合、呼び出し元はすべてのソケット・タイプを受け入れます。ai_protocol の値がゼロの場合、呼び出し元はすべてのプロトコルを受け入れます。hints が NULL ポインターである場合、その動作は、ai_flags、ai_socktype、ai_protocol の各フィールドと ai_family フィールドの AF_UNSPEC の値がゼロである構造を参照しているようになります。

hints 引数が指す ai_flags メンバーは、0 に設定することも、以下の値のいずれか (複数可) のビット単位包含 OR (包含論理和) にすることもできます。

- AI_PASSIVE
- AI_CANONNAME
- AI_NUMERICHOST
- AI_NUMERICSERV
- AI_V4MAPPED
- AI_ALL
- AI_ADDRCONFIG
- AI_EXTFLAGS

AI_PASSIVE ビットが hints 構造体の ai_flags メンバーの中に設定される場合、呼び出し元は、戻されたソケット・アドレス構造体を bind() の呼び出しに使用することを計画します。この場合、nodename 引数が NULL ポインターであれば、ソケット・アドレス構造体の IP アドレス部分は、IPv4 アドレス用の INADDR_ANY または IPv6 アドレス用の IN6ADDR_ANY_INIT に設定されます。AI_PASSIVE ビットが hints 構造体の ai_flags メンバーの中に設定されない場合、戻されたソケット・アドレス構造体は、connect() (接続指向のプロトコル) または connect()、sendto()、または sendmsg() (接続のないプロトコル) のいずれかへの呼び出しの作動可能です。この場合、nodename 引数が NULL ポインターならば、ソケット・アドレス構造体の IP アドレス部分はループバック・アドレスに設定されます。

AI_CANONNAME ビットが hints 構造体の ai_flags メンバーの中に設定された場合、戻りが正常に終了した時に、リンク・リストの先頭の addrinfo 構造体の ai_canonname メンバーは、指定された nodename の正規名を含むヌル終了ストリングを指します。

AI_NUMERICHOST ビットが hints 構造体の ai_flags メンバーの中に設定された場合、非ヌル nodename ストリングは数字ホスト・アドレス・ストリングである必

要があります。そうでないと、エラー・コードの EAI_NONAME が戻されます。このフラグは、どんなタイプのネーム解決サービス (例えば、DNS) も呼び出されることを妨げます。

AI_NUMERICSERV フラグが指定された場合、非ヌル servname ストリングは数値ポート・ストリングである必要があります。そうでない場合は、エラー・コード EAI_NONAME が戻されます。このフラグは、あらゆるタイプのネーム解決サービス (例えば、NIS+) の呼び出しを防止します。

値が AF_INET6 または、システム上で IPv6 がサポートされている時に値が AF_UNSPEC の AF フィールドと共に、AI_V4MAPPED フラグが指定された場合、呼び出し元は IPv4 マップ済み IPv6 アドレスを受け入れます。AI_ALL フラグも指定されておらず、IPv6 アドレスが検索できない場合は、IPv4 アドレスの照会が行われます。不特定の IPv4 アドレスが検索された場合、それらは IPv4 マップ済み IPv6 アドレスとして戻されます。

AF フィールドが AF_INET6 の値を持っていない場合、または AF フィールドが AF_UNSPEC を含むが、IPv6 はシステム上でサポートされていない場合、このフラグは無視されます。AF フィールドが AF_INET6 の値を持っており、AI_ALL が設定されている場合、AI_V4MAPPED フラグも設定されて、呼び出し元はすべてのアドレス (IPv6 および IPv4 マップ済み IPv6 アドレス) を受け入れることを示す必要があります。

システムが IPv6 をサポートする時に AF フィールドが AF_UNSPEC の値を持ち、AI_ALL が設定されている場合に、呼び出し元は、IPv6 アドレスおよび、IPv4 (AI_V4MAPPED が設定されていない場合) または IPv4 マップ済み IPv6 (AI_V4MAPPED が設定されている場合) アドレスのいずれかを受け入れます。まず IPv6 アドレスについて照会が行われ、それが正常に実行されると、IPv6 アドレスが返されます。続いて、IPv4 アドレスの照会が行われ、検出されたアドレスは、IPv4 アドレス (AI_V4MAPPED が設定されていない場合) または IPv4 マップ済み IPv6 アドレス (AI_V4MAPPED が設定されている場合) として戻されます。AF フィールドが、AF_INET6 の値を、またはシステムが IPv6 をサポートする時に AF_UNSPEC の値を持たない場合、フラグは無視されます。

AI_ADDRCONFIG フラグが指定された場合、ノードが少なくとも 1 つの構成された IPv6 ソース・アドレスを持つ場合のみ IPv6 アドレス・レコードの照会が行われます。不特定の IPv4 アドレスが構成されているかどうかにかかわらず、IPv4 アドレス・レコードの照会は常に行われます。この場合、ループバック・アドレスは構成済みのソース・アドレスと同じくらいに有効とは判断されません。

getaddrinfo() が戻したすべての情報は、動的に、addrinfo 構造体、および addrinfo 構造体に指示されたソケット・アドレス構造体と正規ノード名ストリングに割り振られる。この情報をシステムに戻すために、freeaddrinfo() 関数が呼び出されます。

使用上の注意

1. 例えば、呼び出し元が TCP のみを処理して、UDP を処理しない場合に、getaddrinfo() が呼び出された時は、hints 構造体の ai_protocol メンバーが IPPROTO_TCP に設定される必要があります。

2. 呼び出し元が IPV4 のみを処理して IPv6 を処理しない場合に、getaddrinfo() が呼び出された時は、hints 構造体の ai_family メンバーが AF_INET に設定される必要があります。
3. スコープ情報は、IPv6 link-local アドレスのみに関連します。スコープ情報は、link-local アドレスでない IPv4 アドレスおよび IPv6 アドレスの解決では無視されます。

戻り値

正常に実行された場合、getaddrinfo() は、リモート・エントリー・サービス (RES) 引数を介して 0 および 1 つ以上の addrinfo 構造体のリンク・リストを指すポインターを戻します。呼び出し元は、NULL ポインターが表示されるまで、ai_next ポインターに従ってこのリストの中の各 addrinfo 構造体を処理できます。戻されたそれぞれの addrinfo 構造体の中で、ai_family、ai_socktype、および ai_protocol の 3 メンバーは、socket() 関数の呼び出しに対応する引数です。それぞれの addrinfo 構造体の中で、ai_addr メンバーは、その長さが ai_addrlen メンバーに指定される、記入済みのソケット・アドレスを指します。失敗した場合、getaddrinfo() はゼロ以外のエラー・コードを戻します。エラー・コードは、以下の通りです。

エラー・コード
: 説明

EAI_AGAIN
: Node_Name または Service_Name パラメーターが指定した名前は、構成された時間間隔内に解決できなかったか、またはリゾルバー・アドレス・スペースが開始されていません。要求は後で再試行できます。

EAI_BADEXTFLAGS
: 拡張フラグ・パラメーターの設定値が誤っています。

EAI_BADFLAGS
: フラグ・パラメーターの設定値が誤っています。

EAI_FAIL
: リカバリー不能エラーが発生しました。

EAI_FAMILY
: ファミリー・パラメーターの設定値が誤っています。

EAI_MEMORY
: Addr_Info 構造体の獲得を試みている際に、メモリー割り振り失敗が発生しました。

EAI_NONAME
: 以下の状態のいずれかが発生しました。
  1. 名前が指定されたパラメーターを解決しません。名前またはサービス・オペランドの 1 つが指定される必要があります。
  2. 名前要求パラメーターは有効ですが、それはネーム・サーバーにレコードを持っていません。

EAI_SERVICE
: パス済みのサービスが、指定されたソケット・タイプのために認識されませんでした。

EAI_SOCKTYPE
意図されたソケット・タイプは認識されませんでした。

EAI_SYSTEM
システム・エラーが発生しました。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

getclientid() — 呼び出し側アプリケーションの ID の取得

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <socket.h>
#include <types.h>

int getclientid(int domain, struct clientid *clientid);
```

概要

getclientid() 関数は、TCP/IP パーティションで呼び出し側アプリケーションの認識に使用される ID を戻します。givesocket() および takesocket() 呼び出しで *clientid* を使用することができます。

パラメーター
説明

ドメイン
要求されるアドレス・ドメイン。

clientid
埋められる clientid 構造体へのポインター。

以下のように、*clientid* 構造体は、呼び出しによって埋められ、戻されます。

clientid 構造体:

```
struct clientid {
    int domain;
    union  {
     char name[8];
     struct {
       int NameUpper;
       pid_t pid;
     } c_pid;
   } c_name;
   char subtaskname[8];

   struct  {
     char type;
     union  {
        char specific[19];
        struct  {
          char unused[3];
          int SockToken;
     } c_func;
   } c_reserved;
};
```

エレメント
　説明

ドメイン
　clientid 構造体のドメイン・フィールドに戻される入力 *domain* 値。

c_name.name
　アプリケーション・プログラムのパーティション名 (左寄せで、ブランクを入れる)。

subtaskname
　呼び出し側プログラムのタスク ID。

c_reserved
　バイナリー・ゼロを指定。

戻り値

値 0 は成功を表します。値 -1 はエラーを表します。具体的なエラーを errno の値が示します。

errno　説明

EFAULT
　指定された *clientid* パラメーターをそのまま使用すると、呼び出し元のパーティションの外側にあるストレージ、または、呼び出し元が変更できないストレージにアクセスする結果になります。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を返し、errno は EFAULT に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gethostbyaddr() — アドレスによるホスト・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>
extern int h_errno;

struct hostent *gethostbyaddr(const void *address,
                              size_t      address_len,
                              int         domain);
```

概要

gethostbyaddr() 呼び出しは、ネーム・サーバーが存在すれば、ネーム・サーバーを介してホスト・アドレスを解決しようとします。

パラメーター
　説明

address
　ホストのアドレスを含む構造体へのポインター (AF_INET または AF_INET6 では符号なし long 型)。

address_len
address のサイズ (バイト単位)。

ドメイン
サポートされるアドレス・ドメイン (AF_INET または AF_INET6)。

`gethostbyaddr()` 呼び出しは、呼び出しで指定されたホスト・アドレスの **hostent** 構造体へのポインターを戻します。

gethostbyaddr()、および gethostbyname() は、いずれも同じ静的領域を使用して **hostent** 構造体を戻します。この静的領域は、これらの関数のどれかが次に同じスレッドで呼び出されるまでしか有効でありません。

netdb.h インクルード・ファイルは、**hostent** 構造体を定義し、以下のエレメントを含みます。

エレメント
説明

h_addr
ホストのネットワーク・アドレスへのポインター。

h_addrtype
返されるアドレスのタイプ (AF_INET または AF_INET6)。

h_aliases
ホストの代替名のゼロ終了配列。

h_length
アドレスのバイト長。

h_name
ホストの公式名。

以下の関数は、**netdb.h** で定義され、エラー時に *h_errno* 戻り値を参照しようとするとき、マルチスレッドのアプリケーションによって使用されなければなりません。

```
int *__h_errno(void);
```

この関数は、*h_errno* 変数のスレッド固有の値へのポインターを戻します。

戻り値

戻り値は、静的データを示しますが、これは以降の呼び出しで上書きされます。**hostent** 構造体へのポインターは、正常終了を示します。NULL ポインターは、エラーまたはファイルの終わりを示します。

正常に完了しなかった場合、この関数は、エラーを示す *h_errno* を次のように設定します。

エラー・コード
説明

HOST_NOT_FOUND
そのようなホストは認識されていません。

TRY_AGAIN
一時エラー (例えば、サーバーからの応答がない) であり、情報は現在使用可能でないが、後で可能になるかもしれないことを意味します。

NO_RECOVERY
予期しないサーバー障害が発生し、リカバリー方法がありません。

NO_DATA
サーバーは要求と名前を認識したが、使用可能なアドレスがありません。ネーム・サーバーへの別のタイプの要求であれば、応答が戻るかもしれません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返し、h_errno は NO_RECOVERY に、errno は EVSE に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gethostbyname() — 名前によるホスト・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>
extern int h_errno;

struct hostent *gethostbyname(const char *name);
```

概要

gethostbyname() 呼び出しは、ネーム・サーバーが存在すれば、ネーム・サーバーを介してホスト名を解決しようとします。シンボル名を IP アドレスに変換するために呼び出しが行われると、TCP/IP for z/VSE は、ローカル名テーブル (DEFINE NAME によって作成される) を最初に検索します。この検索が失敗した場合、名前は指定された DNS (SET DNSx で設定される) に渡されます。TCP/IP for z/VSE は、応答を受け取るか、すべてのサーバーにポーリングをし終わるまで、DNS1 から始めて各 DNS を試します。応答する最初のサーバーが、要求が成功か失敗かを決定します。ある DNS 内での検索が失敗した場合、デフォルトのドメイン・ストリング (SET DEFAULT_DOMAIN で指定される) が名前に (ピリオドの後に続けて) 付加され、ネーム解決のために最後にその DNS に情報が求められます。

パラメーター
説明

name ホスト名。

gethostbyname() 呼び出しは、呼び出しで指定されたホスト名用の hostent 構造体へのポインターを戻します。

gethostent()、gethostbyaddr()、および gethostbyname() は、いずれも同じ静的領域を使用して hostent 構造体を戻します。この静的領域は、これらの関数のどれかが次に同じスレッドで呼び出されるまでしか有効でありません。

netdb.h インクルード・ファイルは、hostent 構造体を定義し、以下のエレメントを含みます。

エレメント
　説明

h_addr
　ホストのネットワーク・アドレスへのポインター。

h_addrtype
　戻されるアドレスの型。現在は常に AF_INET に設定される。

h_aliases
　ホストの代替名のゼロ終了配列。

h_length
　アドレスのバイト長。

h_name
　ホストの公式名。

以下の関数は、netdb.h で定義され、エラー時に *h_errno* 戻り値を参照しようとするとき、マルチスレッドのアプリケーションによって使用されなければなりません。

```
int *__h_errno(void);
```

戻り値

戻り値は、静的データを示しますが、これは以降の呼び出しで上書きされます。hostent 構造体へのポインターは、正常終了を示します。NULL ポインターは、エラーまたはファイルの終わりを示します。

正常に完了しなかった場合、この関数は、エラーを示す *h_errno* を次のように設定します。

エラー・コード
　説明

HOST_NOT_FOUND
　そのようなホストは認識されていません。

TRY_AGAIN
　一時エラー (例えば、サーバーからの応答がない) であり、情報は現在使用可能でないが、後で可能になるかもしれないことを意味します。

NO_RECOVERY
　予期しないサーバー障害が発生し、リカバリー方法がありません。

NO_DATA
　サーバーは要求と名前を認識したが、使用可能なアドレスがありません。ネーム・サーバーへの別のタイプの要求であれば、応答が戻るかもしれません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返し、h_errno は NO_RECOVERY に、errno は EVSE に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gethostent() — 次のホスト・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

struct hostent *gethostent(void);
```

概要

gethostent() 呼び出しは、既知のホストに関する情報を含むデータ・セットの次の行を読み取ります。

netdb.h 組み込みファイルは、**hostent** 構造体 (以下のエレメントを含む) を定義します。

エレメント
: 説明

h_name
: ホストの公式名。

h_aliases
: ホストの代替名のゼロ終了配列。

h_addrtype
: アドレスのタイプ。

h_length
: アドレスのバイト長。

h_addr
: ホストのネットワーク・アドレスへのポインター。

戻り値

hostent 構造体へのポインターは、正常終了を示します。NULL ポインターは、エラーまたはファイルの終わりを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gethostid() — 現行ホストの固有 ID の取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <unistd.h>

long gethostid(void);
```

概要

gethostid() 呼び出しは、現行ホストの固有な 32 ビット ID を取得します。この値は、デフォルトのホーム IP アドレスです。

戻り値

gethostid() は現行ホストの 32 ビット ID を戻します。この ID はすべてのホスト間で固有のはずです。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 0 を返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gethostname() — ホスト・プロセッサー名の取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <unistd.h>

int gethostname(char *name, size_t namelen);
```

概要

gethostname() 呼び出しは、プログラムが実行中のホスト・プロセッサーの名前を戻します。*namelen* 文字までを name 配列へコピーします。戻された name は、name 配列に十分なスペースがある限り、ヌル終了になります。

パラメーター
: 説明

name
: ホスト名で埋まる文字配列。

namelen
: *name* の長さ。

戻り値

値 0 は成功を表します。値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
: 説明

EFAULT
: *name* および *namelen* を使用すると、呼び出し側のアドレス・スペース内のデータを書き込めない部分へ、アドレスをコピーしようとする結果となります。

EMVSPARM
: 正しくないパラメーターがサービスに渡されたか、関数がインプリメントされていません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EMVSPARM に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getibmopt() - IBM TCP/IP イメージの取得

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <socket.h>


int getibmopt(int cmd, struct ibm_gettcpinfo *bfrp);
```

概要

getibmopt() 呼び出しは、特定の z/VSE システムにインストールされている TCP/IP イメージの数、その状況、バージョン、および名前を返します。この情報に基づき、呼び出し側は setibmopt() 呼び出しを使用して、接続先とする TCP/IP イメージを動的に選択できます。getibmopt() 呼び出しは、オプションです。getibmopt() 呼び出しを使用しない場合は、標準方式に従って、接続先の TCP/IP イメージを判別します。

パラメーター
: 説明

cmd 処理するコマンドを指定する値またはフルワード 2 進数のアドレス。有効な値は 1 のみです。

bfrp ibm_gettcpinfo 構造体へのポインター。

TCP/IP イメージをソケットに設定するには、アプリケーションは、値を以下のように ibm_tpcimage 構造体に設定する必要があります。

エレメント
: 説明

状況 0 は、未確認、検査不要を意味します。現在では、これは意味を持つ唯一の値です。

バージョン
: 0 は、分かっている場合は、戻る時点で設定されるという意味です。

名前 名前は、左そろえされている大文字であり、ブランクが埋め込まれている必要があります。また、アクティブ TCP スタックの名前でなければなりません。

戻り値

正常終了して返された場合、struct ibm_tcpimage バッファーには、最大 8 つのアクティブな TCP/IP イメージの状況、バージョン、および名前が入っています。

エラー・コード
: 説明

EOPNOTSUPP
: この関数はサポートされていない。

EFAULT
: name パラメーターが呼び出し側アドレス・スペース外のアドレスを指定している。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

getnameinfo() - 名前情報の取得

フォーマット

```
#define _OPEN_SYS_SOCK_IPV6
#include <socket.h>
#include <netdb.h>


int getnameinfo(cons struct sockaddr *sa, socklen_t salen,
                char *host, socklen_t hostlen,
                char *serv, socklen_t servlen,
                int flags);
```

概要

getnameinfo() 関数は、ソケット・アドレスをノード名とサービス・ロケーションに変換します。getnameinfo() 関数は、DNS およびシステム特定のデータベースで呼び出し元が提供した IP アドレス、およびポート番号を検索し、呼び出し元の提供するバッファーに両方のテキスト・ストリングを戻します。

sa 引数は、IP アドレスとポート番号を保持する sockaddr_in 構造体 (IPv4 用) または sockaddr_in6 構造体 (IPv6 用) のいずれかを指します。この sockaddr_in6 構造体で表された IPv6 アドレスが link_local アドレスの場合、sockaddr_in6 構造体はゾーン・インデックス値も含みます。salen 引数は、sockaddr_in または sockaddr_in6 構造体の長さを与えます。

ソケット・アドレス構造体が IPv4 マップ済み IPv6 アドレス、または IPv4 互換の IPv6 アドレスを含む場合は、組み込み IPv4 アドレスが抽出され、IPv4 アドレス上で検索が実行されます。

注**:** IPv6 未指定アドレス (「::」) および IPv6 ループバック・アドレス (「::1」) は、IPv4 互換のアドレスではありません。アドレスが IPv6 未指定アドレスの場合、検索は実行されず、EAI_NONAME エラー・コードが戻されます。

IP アドレスに関連付けられたノード名は、ホスト引数によって指し示されるバッファーに戻されます。呼び出し元は、このバッファーのサイズを *hostlen* 引数で提供します。呼び出し元は、*hostlen* または、ヌル・ホスト引数にゼロ値を指定することにより、ノード名を戻さないように指定します。ノード名が見つからない場合、ノード・アドレスの数値フォームが名前の代わりに戻されます。ゾーン・インデックス値が sockaddr_in6 構造体に提供された場合、ノード name%scope 情報形式を使用した、ゾーン・インデックスの数値フォーム、またはゾーン・インデックスに関連付けされたインターフェース名が、戻されるノード名に追加されます。

hostlen 引数に指定されたバッファーのサイズが不十分で、ノード名全体またはノード名とスコープ情報の組み合わせを格納できない場合、*hostlen* に指定された文字数までは、ヌル終了ストリングとしてバッファーにコピーされます。

ポート番号に関連付けられたサービス名は、*serv* 引数によって指定されたバッファー内に戻され、*servlen* 引数がこのバッファーの長さを与えます。呼び出し元は、

servlen または、ヌル *serv* 引数にゼロ値を指定することにより、サービス名を戻さないように指定します。サービス名が見つからない場合、サービス・アドレスの数値 (例えば、サービス・アドレスのポート番号) が名前の代わりに戻されます。

servlen 引数に指定されたバッファーのサイズが不十分で、サービス名全体を格納できない場合、*servlen* 文字までは、ヌル終了ストリングとしてバッファーにコピーされます。

最終引数 *flags* は、この関数のデフォルトのアクションを変更するフラグです。デフォルトでは、ホストの完全修飾ドメイン名 (FQDN) が戻されます。

フラグ・ビット NI_NOFQDN が設定された場合、FQDN のノード名部分のみがローカル・ホストに戻されます。

フラグ・ビット NI_NUMERICHOST が設定された場合、ホスト・アドレスの数値フォームがその名前の代わりに戻されます。

フラグ・ビット NI_NAMEREQD が設定された場合、ホスト名が見つからないとエラーが戻されます。

フラグ・ビット NI_NUMERICSERV が設定された場合、サービス・アドレスの数値フォーム (例えば、サービス・アドレスのポート番号) がその名前の代わりに戻されます。

フラグ・ビット NI_NUMERICSCOPE が設定された場合、スコープ ID の数値フォーム (例えば、ゾーン・インデックス) がその名前の代わりに戻されます。sa 引数が IPv6 アドレスでない場合、このフラグは無視されます。

フラグ・ビット NI_DGRAM が指定される場合、サービスがデータグラム・サービスであることを示しています。また、この指定により、getservbyport() の呼び出しが、デフォルト引数の「tcp」ではなく、2 番目の引数である「udp」で行われるようになります。このフラグは、UDP および TCP への異なるサービスを持ついくつかのポート (例えば、[512,514]) に必要です。

注**:** 多くのコマンドが提供する「-n」フラグのサポートには、3 つの NI_NUMERICxxx フラグが必要です。

戻り値

正常に実行が完了した場合、getnameinfo() は提供されたバッファーにノードおよびサービス名 (それを要求した場合) を戻します。戻される名前は、常にヌル終了ストリングです。getnameinfo() のゼロの戻り値は、正常に実行が完了したことを示します。ゼロ以外の戻り値は失敗を示しています。失敗した場合、以下の値が返される可能性があります。

エラー・コード
: 説明

EAI_AGAIN
: 指定されたホスト・アドレスは、構成された時間間隔内に解決されないか、またはリゾルバー・アドレス・スペースが開始されていません。要求は後で再試行できます。

EAI_BADFLAGS
フラグ・パラメーターの値が誤っています。

EAI_FAIL
リカバリー不能エラーが発生しました。

EAI_FAMILY
アドレス・ファミリーが認識されなかったか、またはアドレスの長さが指定されたファミリーに認識されませんでした。

EAI_MEMORY
メモリー割り振り失敗が発生しました。

EAI_NONAME
名前が提供されたパラメーターを解決しません。次のいずれかが発生しました。

1. NI_NAMEREQD が設定され、ホスト名が見つかりません。
2. ホスト名とサービス名の両方がヌルです。
3. 要求されたアドレスは有効ですが、それはネーム・サーバーにレコードを持っていません。

EAI_OVERFLOW
引数バッファーがオーバーフローしました。ホスト名またはサービス名用に指定したバッファーは、解決済みの名前全体を入れるのに十分ではなく、呼び出し元は切り捨てを行わないことを意味する _EDC_SUSV3=1 を指定しています。

EAI_SYSTEM
リカバリー不能エラーが発生しました。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

getnetbyaddr() — アドレスによるネットワーク・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

struct netent *getnetbyaddr(ip_addr_t net, int type);
```

概要

getnetbyaddr() 呼び出しは、指定されたネットワーク・アドレスを、既知のネットワークに関する情報を含むデータ・セットから検索します。

パラメーター
説明

net ネットワーク・アドレス。

type アドレス・ドメイン。

netent 構造体は、**netdb.h** 組み込みファイルに定義され、以下のエレメントを含みます。

エレメント
　　説明

n_addrtype
　　ネットワーク・アドレスのタイプ。

n_aliases
　　ネットワークの代替名の、NULL ポインターで終了する配列。

n_name
　　ネットワークの公式名。

n_net　ホスト・バイト・オーダーに戻される、ネットワーク番号。

戻り値

netent 構造体へのポインターは、正常終了を示します。NULL ポインターはエラーを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getnetbyname() — 名前によるネットワーク・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

struct netent *getnetbyname(const char *name);
```

概要

getnetbyname() 呼び出しは、指定されたネットワーク名を、既知のネットワークに関する情報を含むデータ・セットから検索します。

パラメーター
　　説明

name　ネットワーク名へのポインター。

netent 構造体は、**netdb.h** 組み込みファイルに定義され、以下のエレメントを含みます。

エレメント
　　説明

n_addrtype
　　ネットワーク・アドレスのタイプ。

n_aliases
　　ネットワークの代替名の、NULL ポインターで終了する配列。

n_name
ネットワークの公式名。

n_net ホスト・バイト・オーダーに戻される、ネットワーク番号。

戻り値

netent 構造体へのポインターは、正常終了を示します。NULL ポインターはエラーを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getnetent() — 次のネットワーク・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

struct netent *getnetent(void);
```

概要

getnetent() 呼び出しは、既知のネットワークに関する情報を含むデータ・セットの次のエントリーを読み取ります。

netent 構造体は、**netdb.h** 組み込みファイルに定義され、以下のエレメントを含みます。

エレメント
説明

n_addrtype
ネットワーク・アドレスのタイプ。

n_aliases
ネットワークの代替名の、NULL ポインターで終了する配列。

n_name
ネットワークの公式名。

n_net ホスト・バイト・オーダーに戻される、ネットワーク番号。

戻り値

netent 構造体へのポインターは、正常終了を示します。NULL ポインターは、エラーまたはファイルの終わりを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getpeername() — ソケットに接続されたピアの名前の取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

int getpeername(int socket, struct sockaddr *name, size_t *namelen);
```

概要

getpeername() 呼び出しは、ソケット記述子 *socket* に接続されたピアの名前を戻します。*namelen* は、*name* で示されるスペースのサイズを示すよう初期設定されなければならず、呼び出しが戻る前に、スペースにコピーされたバイト数に設定されます。ピア名のサイズは、バイト単位で戻されます。アドレスの実際の長さが、供給される *sockaddr* の長さを超える場合、保管されるアドレスは切り捨てられます。構造体 *sockaddr* の *sa_len* フィールドには、切り捨てられていないアドレスの長さが含まれます。

パラメーター
: 説明

socket
: ソケット記述子。

name
: 戻る前に、getpeername() で埋められる接続ソケットの IP アドレス。*name* の正確な形式は、通信が発生するドメインによって決まります。

namelen
: *name* が指すアドレス構造体のサイズ (バイト単位)。

AF_INET6 ドメインのソケット**:** AF_INET6 ソケットでは、アドレスは sockaddr_in6 アドレス構造体に戻されます。sockaddr_in6 構造体は、ヘッダー・ファイル **in.h** の中に定義されます。

戻り値

値 0 は成功を表します。値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
: 説明

EBADF
: *socket* パラメーターが無効ソケット記述子です。

EFAULT
: *name* および *namelen* パラメーターの指定を適用すると、呼び出し側のアドレス・スペースの外側のストレージをアクセスすることになります。

EINVAL
: *namelen* パラメーターが有効な長さではありません。

ENOBUFS
: getpeername() は、ストレージ不足のため、要求を処理できません。

ENOTCONN
: ソケットは、接続状態ではありません。

EOPNOTSUPP
その操作は、ソケット・プロトコルではサポートされません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EBADF に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getprotobyname() — 名前によるプロトコル・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

struct protoent *getprotobyname(const char *name);
```

概要

getprotobyname() 呼び出しは、指定されたプロトコル名を、既知のプロトコルに関する情報を含むデータ・セットから検索します。

パラメーター
説明

name プロトコル名。

protoent 構造体は、**netdb.h** 組み込みファイルで定義され、以下のようなエレメントを持ちます。

エレメント
説明

p_aliases
プロトコルの代替名の、NULL ポインターで終了する配列。

p_name
プロトコルの公式名。

p_proto
プロトコル番号。

戻り値

protoent 構造体へのポインターは、正常終了を示します。NULL ポインターはエラーを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getprotobynumber() — 番号によるプロトコル・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

struct protoent *getprotobynumber(int proto);
```

概要

getprotobynumber() 呼び出しは、指定されたプロトコル番号を、既知のプロトコルに関する情報を含むデータ・セットから検索します。

パラメーター
　説明

proto　プロトコル番号。

protoent 構造体は、**netdb.h** 組み込みファイルで定義され、以下のようなエレメントを持ちます。

エレメント
　説明

p_aliases
　プロトコルの代替名の、NULL ポインターで終了する配列。

p_name
　プロトコルの公式名。

p_proto
　プロトコル番号。

戻り値

protoent 構造体へのポインターは、正常終了を示します。NULL ポインターはエラーを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getprotoent() — 次のプロトコル・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

struct protoent *getprotoent(void);
```

概要

getprotoent() 呼び出しは、既知のプロトコルに関する情報を含むデータ・セットを読み取ります。getprotoent() 呼び出しは、データ・セット内の次のエントリーを指すポインターを返します。

protoent 構造体は、**netdb.h** 組み込みファイルで定義され、以下のようなエレメントを持ちます。

エレメント
　　説明

p_aliases
　　プロトコルの代替名の、NULL ポインターで終了する配列。

p_name
　　プロトコルの公式名。

p_proto
　　プロトコル番号。

戻り値

protoent 構造体へのポインターは、正常終了を示します。NULL ポインターは、エラーまたはファイルの終わりを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getservbyname() — 名前によるサービス・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

struct servent *getservbyname(const char *name, const char *proto);
```

概要

`getservbyname()` 呼び出しは、指定されたサービス名およびプロトコル名に一致する最初のエントリーを、既知のサービスに関する情報を含むデータ・セットから検索します。*proto* が NULL の場合は、サービス名のみが一致する必要があります。

パラメーター
　　説明

name　サービス名。

proto　プロトコル名。

servent 構造体は、**netdb.h** 組み込みファイルで定義され、以下のようなエレメントを持っています。

エレメント
　　説明

s_aliases
　　サービスの代替名の、NULL ポインターで終了する配列。

s_name
　　サービスの公式名。

s_port サービスのポート番号。

s_proto
サービスに接続するのに必要なプロトコル。

戻り値

servent 構造体へのポインターは、正常終了を示します。NULL ポインターはエラーを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getservbyport() — ポートによるサービス・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

struct servent *getservbyport(int port, const char *proto);
```

概要

getservbyport() 呼び出しは、指定されたポート番号およびプロトコル名に一致する最初のエントリーを、サービスに関する情報を含むデータ・セットから検索します。*proto* が NULL の場合は、ポート番号のみが一致しなければなりません。

パラメーター
説明

port ポート番号。

proto プロトコル名。

servent 構造体は、**netdb.h** 組み込みファイルで定義され、以下のようなエレメントを持っています。

エレメント
説明

s_aliases
サービスの代替名の、NULL ポインターで終了する配列。

s_name
サービスの公式名。

s_port サービスのポート番号。

s_proto
サービスに接続するのに必要なプロトコル。

戻り値

servent 構造体へのポインターは、正常終了を示します。NULL ポインターはエラーを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getservent() — 次のサービス・エントリーの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

struct servent *getservent(void);
```

概要

getservent() 呼び出しは、データ・セットの次の行を読み取り、サービスに関する情報を含む、データ・セット内の次のエントリーを指すポインターを返します。

servent 構造体は、**netdb.h** 組み込みファイルで定義され、以下のようなエレメントを持っています。

エレメント
: 説明

s_aliases
: サービスの代替名の、NULL ポインターで終了する配列。

s_name
: サービスの公式名。

s_port サービスのポート番号。

s_proto
: サービスに接続するのに必要なプロトコル。

戻り値

servent 構造体へのポインターは、正常終了を示します。NULL ポインターは、エラーまたはファイルの終わりを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getsockname() — ソケット名の取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

int getsockname(int socket, struct sockaddr *name, size_t *namelen);
```

概要

getsockname() 呼び出しは、*socket* パラメーターで指定されたソケットの現行名を、*name* パラメーターで示された構造体に保管します。バインドされているソケットへのアドレスを戻します。ソケットがアドレスにバインドされていない場合は、呼び出しは、ファミリーを設定し、構造体の残りの部分はゼロに設定して戻ります。例えば、インターネット・ドメイン内のバインドされていないソケットの場合、name は **sockaddr_in** 構造体を指し、構造体の *sin_family* フィールドは AF_INET に設定され、その他のすべてのフィールドはゼロに設定されます。

アドレスの実際の長さが、供給される *sockaddr* の長さを超える場合、保管されるアドレスは切り捨てられます。構造体 *sockaddr* の *sa_len* フィールドには、切り捨てられていないアドレスの長さが含まれます。

パラメーター
: 説明

socket
: ソケット記述子。

name
: getsockname() が *socket* の名前をコピーするバッファーのアドレス。

namelen
: 最初は *name* が指すストレージのサイズ (バイト) を含む整数を示していなければなりません。戻るときには、その整数に、*name* が指すストレージに戻されたデータのサイズが入ります。

AF_INET6 ドメインのソケット**:** AF_INET6 ソケットでは、アドレスは sockaddr_6 アドレス構造体に戻されます。sockaddr_in6 構造体は、ヘッダー・ファイル **in.h** の中に定義されます。

getsockname() 呼び出しは、通常、ソケットが暗黙的にポートへバインドされた後、ソケットへ割り当てられたポートを見つけるのに使用されます。例えば、アプリケーションでは、bind() を前に呼び出さなくても、connect() を呼び出すことができます。この場合、connect() 呼び出しは、ポートをソケットに割り当て、必要なバインディングを実行します。この割り当てを、getsockname() を呼び出して見つけることができます。

戻り値

値 0 は成功を表します。値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
: 説明

EBADF
: *socket* パラメーターが無効ソケット記述子です。

EFAULT
: *name* および *namelen* パラメーターの指定を適用すると、呼び出し側のアドレス・スペースの外側のストレージをアクセスすることになります。

ENOBUFS
: getsockname() は、ストレージ不足のため、要求を処理できません。

ENOTCONN
ソケットは、接続状態ではありません。

EOPNOTSUPP
その操作は、ソケット・プロトコルではサポートされません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

getsockopt() — ソケットに関連したオプションの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

int getsockopt(int     socket,
               int     level,
               int     option_name,
               void   *option_value,
               size_t *option_len);
```

概要

getsockopt() 呼び出しは、ソケットに関連するオプションを取得します。すべてのオプションが、すべてのアドレス・ファミリーによってサポートされるわけではありません。詳細は、各オプションを参照してください。オプションは、複数のプロトコル・レベルで指定できます。最上位のソケット・レベルには必ず存在します。

パラメーター
説明

socket ソケット記述子。

level オプションを設定するレベル。

option_name
指定ソケット・オプションの名前。

option_value
オプション・データへのポインター。

option_len
オプション・データの長さへのポインター。

ソケット・オプションを操作する際、オプションがどのレベルにあるのかとオプション名を指定する必要があります。ソケット・レベルのオプションを操作するため、*level* パラメーターを **socket.h** に定義されているように SOL_SOCKET に設定する必要があります。 IPv4 または IPv6 レベルでオプションを操作するには、level パラメーターを、**socket.h** の定義に従って IPPROTO_IP に設定するか、または **in.h** の定義に従って IPPROTO_IPV6 に設定する必要があります。その他のレベル (TCP レベルなど) のオプションを操作するには、オプションを制御しているプロトコルの該当プロトコル番号を提供します。 getprotobyname() 呼び出しを使用して、名前を指定されたプロトコルのプロトコル番号を戻すことができます。

option_value および *option_len* パラメーターが、特定の取得コマンドで使用されるデータを戻すために使用されます。*option_value* パラメーターは、取得コマンドによって要求されるデータを受信するためのバッファーを指します。*option_len* パラメーターは、*option_value* パラメーターで示されるバッファーのサイズを指します。それは、`getsockopt()` を呼び出す前に、バッファーのサイズに初期設定しておかなければなりません。戻り時に、戻されたデータの実際のサイズに設定されます。

SO_LINGER、SO_RCVTIMEO、SO_SNDTIMEO を除くすべてのソケット・レベルのオプションでは、*option_value* が整数を指していて、*option_len* には整数のサイズが設定されていると予期されます。整数がゼロ以外の場合には、オプションは使用可能です。ゼロの場合は、オプションは使用不可です。SO_LINGER オプションの場合は、*option_value* は、**socket.h** で定義されている **linger** 構造体を示していると想定されます。この構造体の定義を以下の例に示します。

```
struct  linger
{
       int     l_onoff;                /* option on/off */
       int     l_linger;               /* linger time */
};
```

l_onoff フィールドは、SO_LINGER オプションが使用不可にされる場合、ゼロに設定されます。ゼロ以外の値の場合、オプションは使用可能になります。*l_linger* フィールドは、クローズ時に残っている時間の長さを指定します。

以下のようなオプションが、ソケット・レベルで認識されます。

オプション
　説明

SO_ACCEPTCONN
　ソケットは listen() 呼び出しを行いました。

SO_BROADCAST
　メッセージをブロードキャストする機能を切り替えます。このオプションを使用可能にすると、宛先に指定したインターフェースがパケットのブロードキャストをサポートするならば、アプリケーションは、ブロードキャスト・メッセージをソケット上で送ることができます。このオプションは、ストリーム・ソケットには何の意味もありません。このオプションは、AF_INET ドメインの場合にのみ有効です。

SO_DEBUG
　デバッグ情報が記録されているかどうかを報告します。このオプションは int 値を保管します。

SO_ERROR
　ソケット上の保留中のエラーを戻し、エラー状況をクリアします。
　SO_ERROR を使用すると、接続されたデータグラム・ソケット上の非同期エラー、またはその他の非同期エラー (ソケット呼び出しの 1 つで明示的に戻されないエラー) をチェックすることができます。

SO_KEEPALIVE
　ストリーム・ソケットの TCP キープアライブ・メカニズムを切り替えます。活動化されていると、キープアライブ・メカニズムによって、パケットが定期的に異なるアイドル接続に送信されます。リモート TCP がパケット、またはパケットの再送に応答しない場合は、接続は、エラー

ETIMEDOUT で終了します。そのソケットへ書き込むプロセスは、SIGPIPE シグナルで通知されます。このオプションは int 値を保管します。このオプションは、AF_INET および AF_INET6 ドメインの場合にのみ有効です。

SO_LINGER
データが存在している場合、クローズ時にすぐに戻りません。このオプションが使用可能にされている場合、close() が呼び出されたときに未送信のデータが存在すると、データが転送されるまで、または接続がタイムアウトになるまで、close() 呼び出し中は呼び出し側アプリケーションがブロックされます。このオプションが使用不可にされている場合、TCP/IP アドレス・スペースは、データを送信しようとするのを待ちます。データ転送は通常は成功しますが、TCP/IP アドレス・スペースがデータを送信しようとするのを待つ時間は限られているので、成功が保証されるわけではありません。close() 呼び出しは、呼び出し元をブロックしないで、戻ります。このオプションは、ストリーム・ソケットにのみ意味を持ちます。

SO_OOBINLINE
アウト・オブ・バンドのデータの受信を切り替えます。このオプションが使用可能な場合、受信したアウト・オブ・バンドのデータは、通常のデータ入力キューに入れられます。その後、このデータを、recv()、recvfrom()、および recvmsg() 呼び出しで使用することができます。その際に MSG_OOB フラグを指定する必要はありません。このオプションが使用不可の場合、受信したアウト・オブ・バンドのデータは、優先順位データ入力キューに入れられます。MSG_OOB フラグを指定した場合のみ、recv()、recvfrom()、および recvmsg() 呼び出しでこのデータを使用することができます。このオプションは、ストリーム・ソケットにのみ意味を持ちます。

SO_RCVTIMEO
入力関数がその完了まで待つ時間とともにタイムアウト値を報告します。追加データを受信することなくこの時間の間ブロックしていた受信操作は、部分的なカウントで戻るか、受信したデータがない場合は errno を EWOULDBLOCK に設定して戻ります。このオプションのデフォルトはゼロです。これは、受信操作がタイムアウトしないことを意味します。

SO_REUSEADDR
ローカル・アドレスの再使用を切り替えます。このオプションを使用可能にすると、既に使用中のローカル・アドレスをバインドすることができます。SO_REUSEADDR は、bind() 呼び出しで使用される通常のアルゴリズムを変更する。ローカル・アドレスおよびポートに、同じ外部アドレスおよびポートがないことを確認するために、システムでは接続時間に検査が行われます。関連が既に存在している場合には、エラー EADDRINUSE が戻されます。SO_REUSEADDR オプションがアクティブな場合は、呼び出しごとに異なるローカル IP アドレスが使用され、かつ、ワイルドカード・アドレス INADDR_ANY がポートごとに 1 回のみ使用されている限り、サーバーは同じポートを複数回 bind() できます。このオプションは、AF_INET および AF_INET6 ドメインの場合にのみ有効です。

SO_SNDBUF
送信バッファー・サイズ情報を報告します。このオプションは int 値を保管します。

SO_SNDTIMEO
データ送信を防止するフロー制御が原因で出力関数がブロックする時間を指定する、タイムアウト値を報告します。この時間の間ブロックされた送信操作は、部分的なカウントで戻るか、送信されたデータがない場合は errno を EWOULDBLOCK に設定して戻ります。このオプションのデフォルトはゼロです。これは、送信操作がタイムアウトにならないことを意味します。

SO_TYPE
このオプションは、ソケットのタイプを戻します。戻り時に、option_value で指定される整数が、SOCK_STREAM または SOCK_DGRAM に設定されます。このオプションは、AF_INET および AF_INET6 ドメインの場合に有効です。

以下のオプションが、IPv4 レベルで認識されます。

オプション
説明

IP_MULTICAST_IF
アウトバウンド・マルチキャスト・データグラムの送信に使用されるインターフェース IP アドレスを返します。IP アドレスは struct in_addr を使用して返されます。

IP_MULTICAST_LOOP
ループバックが使用可能か使用不可かを判別します。ループバック標識が u_char として返されます。0 はループバックが使用不可であることを示し、1 は使用可能であることを示します。

IP_MULTICAST_TTL
発信マルチキャスト・データグラムの IP データグラム存続時間 (ホップ) を返します。TTL 値が u_char として返されます。

以下のオプションが、IPv6 レベルで認識されます。

オプション
説明

IPV6_MULTICAST_HOPS
アウトバウンド・マルチキャスト・データグラムのホップ限界値を返します。ホップ限界値は int として返されます。

IPV6_MULTICAST_IF
アウトバウンド・マルチキャスト・データグラムの送信に使用されたインターフェースのインターフェース索引を返します。インターフェース索引は、struct u_int を使用して返されます。

IPV6_MULTICAST_LOOP
発信マルチキャスト・パケットのループバックが使用可能か使用不可かを判別します。ループバック標識は u_int として返されます。0 は関数が使用不可であることを示し、1 は使用可能であることを示します。

IPV6_UNICAST_HOPS
アウトバウンド・ユニキャスト・データグラムのホップ限界値を返します。ホップ限界値は int として返されます。

IPV6_V6ONLY
ソケットが IPv6 コミュニケーションのみに制限されているかどうかを判別します。オプション値は int として返されます。ゼロ以外の値は、オプションが使用可能 (ソケットは IPv6 コミュニケーションのみに使用可能) であることを示します。0 は、オプションが使用不可であることを示します。

戻り値

値 0 は成功を表します。値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
説明

EBADF
socket パラメーターが無効ソケット記述子です。

EFAULT
option_value および *option_len* パラメーターを使用すると、呼び出し元のアドレス・スペースの外側のストレージにアクセスすることになります。

EINVAL
指定されたオプションは、指定されたソケット・レベルでは無効です。

ENOBUFS
メッセージの送信にバッファー・スペースを使用することができません。

ENOPROTOOPT
option_name パラメーターが認識されていないか、または *level* パラメーターが SOL_SOCKET ではありません。

ENOSYS
関数がインプリメントされていません。まだ使用可能になっていない関数を使用しようとしました。

EOPNOTSUPP
その操作は、ソケット・プロトコルではサポートされていません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは値 -1 を戻し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

スタックの特性

TCP/IP for z/VSE ではオプション SO_LINGER のみがサポートされています。

例

以下に示すのは、getsockopt() 呼び出しの例です。setsockopt() 呼び出しオプションを設定する方法の例は、 211 ページの『setsockopt() — ソケットに関連したオプションの設定』を参照してください。

```
int rc;
int s;
int option_value;
int option_len;
```

```
struct linger l;

:
/* Do I linger on close? */
option_len = sizeof(l);
rc = getsockopt( s,
                    SOL_SOCKET,
                    SO_LINGER,
                    (char *)&l,
                    &option_len);
if (rc == 0)
{
    if (option_len == sizeof(l))
    {
         if (l.l_onoff)
            /* yes I linger */
         else
            /* no I do not  */
    }
}
```

givesocket() — 指定したソケットを使用可能にする

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <socket.h>

int givesocket(int d,struct clientid *clientid);
```

概要

givesocket() 呼び出しは、指定されたソケットを別のプログラムから発行される takesocket() 呼び出しで使用できるようにします。任意のソケットを指定できます。通常、givesocket() はマスター・プログラムによって使用されます。マスター・プログラムが accept() によってソケットを取得してアプリケーション・プログラムに渡し、アプリケーション・プログラムが一度に 1 つのソケットを扱うというのが標準的な処理です。

パラメーター
: 説明

ソケット
: 別のアプリケーションに与えられるソケットの記述子。

clientid
: ソケットが与えられるプログラムを指定するクライアント ID 構造体へのポインター。

ソケットを渡すには、受け渡し元のプログラムでまず、以下のように埋められたクライアント ID 構造体を指定して givesocket() を呼び出します。

clientid 構造体:

```
 struct clientid {
    int domain;
    union  {
      char name[8];
      struct {
        int NameUpper;
        pid_t pid;
      } c_pid;
```

```
    } c_name;
    char subtaskname[8];

    struct  {
      char type;
      union  {
         char specific[19];
         struct  {
           char unused[3];
           int SockToken;
         } c_close;
      } c_func;
    } c_reserved;
 };
```

エレメントの説明

エレメント
: 説明

ドメイン
: 入力ソケット記述子のドメイン。

c_name.name
: *clientid* が `getclientid()` 呼び出しで設定された場合は、*c_name.name* は、以下のようにすることができます。

- アプリケーション・プログラムのパーティション名 (左寄せで、ブランクを入れる) に設定する。アプリケーション・プログラムは、マスター・プログラムと同じパーティションで実行できます。その場合、このフィールドはマスター・プログラムのパーティションに設定されます。
- ブランクに設定する。そうすると、任意の z/VSE パーティションがソケットを取得できます。

subtaskname
: *clientid* が `getclientid()` 呼び出しで設定された場合は、`subtaskname` は、以下のようにすることができます。

- 引き取り手のタスク ID に設定する。これと、*c_name.name* 値によって、この *c_name.name* および *subtaskname* を持つ 1 つのプロセスだけがソケットを得るようにすることができます。
- ブランクに設定する。*c_name.name* が値を持ち、かつ *subtaskname* がブランクの場合、その *c_name.name* のどのタスクもソケットを得ることができます。
- *c_name_name* がブランクに設定されている場合、*subtaskname* パラメーターはブランクに設定されます。

c_reserved.type
: `SO_CLOSE` に設定されると、ソケットが自動的に `givesocket()` によってクローズされ、ソケットを識別する固有のトークンが *c_close.SockToken* に戻されることを示します。ソケット記述子の代わりに *c_close.SockToken* が `takesocket()` への入力として使用されるよう、受け取り側プログラムへ渡されるべきです。今クローズされたソケット記述子は、`takesocket()` が呼び出されるまでに再利用される可能性があるので、`takesocket()` には *c_close.SockToken* が使用されるべきです。

c_close.SockToken

c_reserved.type が SO_CLOSE に設定されているときにソケット記述子に代わりに、*givesocket* によって戻され、takesocket() への入力として使用される、ソケットを一意的に識別するトークン。

c_reserved

givesocket() によってソケットの自動クローズが行われない場合は、バイナリー・ゼロを指定します。

givesocket/takesocket での名前とサブタスク名の使用

1. 受け渡し元のプログラムは、getclientid()を呼び出して、クライアント ID を取得する。受け渡し元のプログラムは givesocket() を呼び出して、takesocket() 呼び出しでソケットが使用できるようにする。受け渡し元のプログラムはそのクライアント ID を、渡されるソケットの記述子と一緒に、受け取り側プログラムのスタートアップ・パラメーター・リストを介して、受け取り側のプログラムに渡す。
2. 受け取り側プログラムは、受け渡し元のプログラムのクライアント ID およびソケット記述子を指定して、takesocket() を呼び出す。
3. 受け取り側プログラムがソケットを受け取るのを待つ。受け渡し元プログラムは、select() を使用して、受け渡されたソケットで例外条件をテストする。select() が、例外条件が保留中であると報告すると、受け渡し元プログラムは、close() を呼び出して、受け渡されたソケットを解放する。
4. 保留中の例外条件が示される前に、受け渡し元のプログラムがソケットをクローズした場合、その接続はすぐにリセットされ、受け取り側プログラムの takesocket() 呼び出しは正常終了しない。受け渡されたソケットでの close() 呼び出し以外の呼び出しは -1 を戻し、errno を EBADF に設定する。

注: 後方互換性のため、クライアント ID は、ターゲットのプログラムが getclientid() を呼び出して取得した、struct クライアント ID 構造体を指すことができます。この場合、ターゲット・プログラムのみがソケットを得られ、ターゲット・プログラムのパーティション内にある他のプログラムは得られません。

戻り値

値 0 は成功を表します。値 -1 はエラーを表します。具体的なエラーを errno の値が示します。

エラー・コード

説明

EBADF

d パラメーターが無効ソケット記述子です。ソケットはすでに受け渡されています。

EFAULT

指定された *clientid* パラメーターをそのまま使用すると、呼び出し元のパーティションの外側にあるストレージにアクセスする結果になります。

EINVAL

clientid パラメーターが有効なクライアント ID を指定していないか、または、*clientid* ドメインが入力ソケット記述子の *domain* と一致していません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gsk_free_memory() — SSL 用に割り振られたメモリーの解放

フォーマット

```
#include <gskssl.h>

void gsk_free_memory(void *pointer,
                     void *future_use);
```

概要

gsk_free_memory() は、SSL 用に割り振られたメモリーを解放します。

注: gsk_get_dn_by_label() 呼び出しによってヌル終了ストリングで返された識別名は、gsk_free_memory() を使用して解放する必要があります。

パラメーター

説明

pointer

前の SSL 関数呼び出しからアプリケーションに戻された、解放するメモリーのアドレス。

future_use

将来の SSL での使用のため予約済み。

gsk_get_cipher_info() — 暗号関連情報の照会

フォーマット

```
#include <gskssl.h>

int gsk_get_cipher_info(int level,
                        gsk_sec_level *sec_level,
                        void *Reserved_for_future_use);
```

概要

SSL の暗号関連情報を照会します。gsk_get_cipher_info() は、システムがサポートできる暗号化レベルを判別し、SSL が使用できる暗号仕様のリストを戻します。これにより、アプリケーションはその実行時に、インストール済みアプリケーションが要求できる SSL 暗号化のレベルを判別することができます。この関数は、世界各地で稼働するシステム上で実行するプログラムでの使用に役立ちます。

gsk_get_cipher_info() を使用して、gsk_secure_soc_init() によって使用される *gsk_soc_init_data* 構造体の暗号仕様に指定できる、有効な値を判別できます。

パラメーター

説明

level 戻される暗号情報のタイプを決定します。GSK_LOW_SECURITY または GSK_HIGH_SECURITY のいずれかを指定してください。
GSK_LOW_SECURITY を指定すると、米国/カナダ外で使用可能な暗号情

報のみが戻されます。GSK_HIGH_SECURITY を指定すると、米国/カナダ外で使用可能な暗号情報と米国/カナダ国内用の暗号情報が戻されます。GSK_LOW_SECURITY は、米国およびカナダ以外の、強力な暗号機能が使用できない場所にあるシステムとの SSL 通信をセットアップする際に有用です。

sec_level
gsk_sec_level 構造体へのポインター。

Reserved_for_future_use
将来の SSL での使用のため予約済み。

gsk_sec_level 構造体は、*gskssl.h* ヘッダー・ファイル内に次のように定義されています。

```
typedef struct _gsk_sec_level {
   int  version;                /* Output - SSL toolkit version     */
   char v3cipher_specs [64];    /* Output - The sslv3 cipher specs  */
   char v2cipher_specs [32];    /* Output - The sslv2 cipher specs  */
   int  security_level;         /* Output - initially one of        */
                                /*      GSK_SEC_LEVEL_US,           */
                                /*      GSK_SEC_LEVEL_EXPORT,       */
                                /*      GSK_SEC_LEVEL_EXPORT_FR     */
} gsk_sec_level;
```

gsk_sec_level 構造体は、システムで使用可能な暗号のレベルについての情報を指定します。アプリケーションは、この構造体用に、必要なメモリーを割り振らなければなりません。成功すると、この構造体の内容が設定されて戻ります。

戻り値

gsk_get_cipher_info() 呼び出しは、整数を戻します。0 以上の値は、正常終了を表します。負の値は、エラーを表します。

GSK_ERROR_IO が戻された場合、一般入出力エラーが発生したことを意味し、errno の値によって具体的なエラーが示されます。

注: この操作中に errno が変わることがあります。しかし、SSL インターフェースが明示的に errno を使用することはなく、エラー原因を判別するために errno が使用されることもありません。戻り値が、SSL API からのエラーを示す唯一の指標です。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の SSL 関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 GSK_ERROR_UNSUPPORTED を返し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gsk_get_dn_by_label() — ラベルに基づく識別名の取得

フォーマット

```
#include <gskssl.h>

char * gsk_get_dn_by_label(char *label);
```

概要

ある鍵の完全な識別名を、その鍵のラベルに基づいて戻します。gsk_initialize() ルーチンを呼び出しておかなければ、gsk_get_dn_by_label() ルーチンを呼び出すことはできません。gsk_secure_soc_init() 呼び出しに使用される gsk_soc_init_data 構造体の DName フィールドに、この値を使用できます。

注: ヌル終了ストリングで返された識別名は、gsk_free_memory() を使用して解放する必要があります。

パラメーター
　説明

ラベル *(label)*
　鍵のラベルを含むヌル終了文字ストリングを指定します。

戻り値

gsk_get_dn_by_label() 呼び出しは、正常終了すると、識別名へのポインターを戻します。指定されたラベルの検出中にエラーが発生すると、ヌル値が戻されます。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の SSL 関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 NULL を返し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gsk_initialize() — SSL 環境の初期設定

フォーマット

```
#include <gskssl.h>

int gsk_initialize(gsk_init_data *init_data);
```

概要

現行パーティションについて、全般的な SSL 環境を設定します。gsk_initialize() が正常に完了すると、アプリケーションは SSL インターフェースを呼び出すことができ、セキュア・ソケット接続の作成と使用を開始できます。

注: gsk_initialize() を複数回呼び出すことができるのは、次に gsk_initialize() を呼び出す前に、gsk_uninitialize() の呼び出しによって既存 SSL 環境がクリーンアップされている場合に限ります。

パラメーター
　説明

init_data
　gsk_init_data 構造体へのポインター。

gsk_init_data 構造体は、*gskssl.h* ヘッダー・ファイル内に次のように定義されています。

```
typedef struct _gsk_init_data { /* Basic gsk SSL Toolkit
                                 * initialization data
                                 */
   char * sec_types;              /* Security protocol choice       */
```

```
                                    /* (SSLV2|SSLV3|...|ALL          */
   char * keyring;                  /* Keyring file name             */
                                    /* Default roots used when NULL  */
   char * keyring_pw;               /* Keyring password              */
                                    /* Ignored when keyring=NULL     */
   char * keyring_stash;
   long V2_session_timeout;        /* Number of seconds for SSLV2    */
                                   /* session data to time out. 0-100 */
   long V3_session_timeout;        /* Number of seconds for SSLV3    */
                                   /* session data to time out.      */
                                   /* 0-86400 (1 day)                */
   char * LDAP_server;             /* Name or IP address of X500 host */
   int    LDAP_port;               /* Port number of X500 host       */
   char * LDAP_user;               /* User name for X500 host        */
   char * LDAP_password;           /* Password of X500 host          */
   gsk_ca_roots LDAP_CA_roots;     /* Which CA roots to use          */
   gsk_auth_type auth_type;        /* Client authentication type     */
} gsk_init_data;
```

sec_types フィールドは、使用されるセキュリティー・プロトコルを指定するヌル終了文字ストリングを指定します。

注: SSLV2 は、現在 VSE 下では使用されません。

keyring フィールドは、鍵および証明書用に使用されるサブライブラリー (フォーマット: "lib.sublib") を指定するヌル終了文字ストリングを指定します。

keyring_pw フィールドは、現在 VSE 下では使用されません。

keyring_stash フィールドは、現在 VSE 下では使用されません。

V2_session_timeout フィールドは、現在 VSE 下では使用されません。

V3_session_timeout フィールドは、SSLV3 セッション ID の有効期限を表す秒数を指定します。範囲は 0 から 86400 秒 (1 日) です。

LDAP_server フィールドは、現在 VSE 下では使用されません。

LDAP_port フィールドは、現在 VSE 下では使用されません。

LDAP_user フィールドは、現在 VSE 下では使用されません。

LDAP_password フィールドは、現在 VSE 下では使用されません。

LDAP_CA_roots フィールドは、証明書の検査に使用する CA ルートを指定します。サポートされる値は、GSK_CA_ROOTS_LOCAL_ONLY および GSK_CA_ROOTS_LOCAL_AND_X500 です。

auth_type フィールドは、クライアントの証明書を検証するのに使用する方式を指定します。このフィールドは、*LDAP_CA_roots* フィールドが GSK_CA_ROOTS_LOCAL_AND_X500 に設定されている場合にのみ使用されます。サポートされる値は、GSK_CLIENT_AUTH_LOCAL、GSK_CLIENT_AUTH_STRONG_OVER_SSL、 GSK_ CLIENT_AUTH_STRONG、および GSK_CLIENT_AUTH_PASSTHRU です。

注: *gsk_init_data* 構造体は、それが指すデータと共に、アプリケーションが SSL を利用する間ずっとアクセス可能のままにするべきです。特に、*gsk_init_data* 構造体

の中のポインターは、アプリケーションによって解放されたストレージまたは呼び出しスタックにあるストレージを指すべきではありません。

戻り値

gsk_initialize() 呼び出しは整数を戻します。値 GSK_INITIALIZE_OK は、SSL 初期設定が成功したことを表します。

GSK_ERROR_IO が戻された場合、一般入出力エラーが発生したことを意味し、errno の値によって具体的なエラーが示されます。

注: この操作中に errno が変わることがあります。しかし、SSL インターフェースが明示的に errno を使用することはなく、エラー原因を判別するために errno が使用されることもありません。戻り値が、SSL API からのエラーを示す唯一の指標です。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の SSL 関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 GSK_ERROR_UNSUPPORTED を返し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gsk_secure_soc_close() — セキュア・ソケット接続のクローズ

フォーマット

```
#include <gskssl.h>

void gsk_secure_soc_close(gsk_soc_data *user_socket);
```

概要

gsk_secure_soc_close() 関数は、セキュア・ソケット接続を終了し、そのセキュア・ソケット接続用のすべての SSL リソースを解放します。

注:

1. gsk_secure_soc_close() を呼び出さないと、*user_socket* パラメーターが参照するストレージは解放されません。
2. ユーザー・アプリケーションは、どのソケット API がオープンしたものでも、すべてのソケット記述子をクローズする必要があります。
   gsk_secure_soc_close() では、オープンしたソケット記述子はクローズされません。

パラメーター
　説明

user_socket
　gsk_soc_data 構造体へのポインター。

戻り値

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の SSL 関数をインプリメントしていない場合は、メッセージ EDCV001I または EDCT002I が出されます。

gsk_secure_soc_init() — セキュア・ソケット接続用データ域の初期設定

フォーマット

```
#include <gskssl.h>

gsk_soc_data * gsk_secure_soc_init(gsk_soc_init_data *soc_init_data);
```

概要

gsk_secure_soc_init() 関数は、SSL がセキュア・ソケット接続を開始または受け入れるために必要なデータ域を初期設定します。gsk_secure_soc_init() が正常終了すると、アプリケーションにハンドルが戻されます。それ以降、このセキュア・ソケット接続を使用する他の呼び出しは、このハンドルを使用できます。

この呼び出し中に、*gsk_soc_init_data* 構造体に指定された入力に基づいて、完全な SSL ハンドシェークが実行されます。SSL ハンドシェークの手順は SSL によって実行されますが、SSL ハンドシェーク中の SSL データの転送と、それに続くすべての読み取り/書き込み操作に必要なルーチンは、アプリケーションが提供する必要があります。

注: それらのルーチンは、fetchep() で生成される外部エントリー・ポイントとして提供される必要があります。

パラメーター
: 説明

soc_init_data
: *gsk_soc_init_data* 構造体へのポインター。

gsk_soc_init_data 構造体は、*gskssl.h* ヘッダー・ファイル内に次のように定義されています。

```
typedef struct _gsk_soc_init_data {
   int    fd;                     /* file descriptor                */
   gsk_handshake hs_type;         /* client or server handshake     */
   char * DName;                  /* keyring entry Distinguished    */
                                  /* name. When NULL the default    */
                                  /* keyring entry is used          */
   char * sec_type;               /* Type of security protocol used */
                                  /* to protect this socket         */
   char * cipher_specs;           /* SSLV2 cipher specs preference  */
   char * v3cipher_specs;         /* SSLV3 cipher specs preference  */
                                  /* and order                      */
   int  (* skread)                /* User supplied READ function ptr */
        (int fd, void * buffer, int num_bytes);
   int  (* skwrite)              /* User supplied WRITE function ptr */
        (int fd, void * buffer, int num_bytes);
   unsigned char cipherSelected[3];   /* Cipher Spec used           */
   unsigned char v3cipherSelected[2]; /* Cipher Spec used           */
   int   failureReasonCode;       /* failure reason code            */
   gsk_cert_info * cert_info;     /* This information is read from  */
                                  /* from the client certificate    */
                                  /* when client authentication is  */
                                  /* enabled                        */
   gsk_init_data * gsk_data;
} gsk_soc_init_data;
```

gsk_soc_init_data 構造体は、セキュア・ソケット接続に関する特性についての情報を指定します。さらに、SSL はこの構造体を使用して、確立された後のセキュア・ソケット接続に関する情報を戻します。

fd フィールドは、この接続のソケット記述子を指定します。ソケット記述子は、*skread* フィールドと *skwrite* フィールドに指定されたアプリケーション・ルーチンに渡されます。アプリケーションが提供するこれらのルーチンは、ソケット記述子を使用して、SSL データの必要な読み取り/書き込みを実行します。

注: `gsk_secure_soc_init()` を呼び出す前に、ソケットの作成、オープン、接続が完了していなければなりません。これは、クライアントは `gsk_secure_soc_init()` を呼び出す前に、`socket()` および `connect()` 呼び出しを実行する必要があることを意味します。サーバーの場合は、`gsk_secure_soc_init()` を呼び出す前に、サーバーが `socket()`、`bind()`、`listen()`、および `accept()` 呼び出しを実行する必要があることを意味します。

hs_type フィールドは、どのように SSL ハンドシェークを実行するのかを指定します。以下の値がサポートされます。

- 認証ありのクライアントとして SSL ハンドシェークを実行する場合、GSK_AS_CLIENT を指定します。
- サーバーとして SSL ハンドシェークを実行する場合、GSK_AS_SERVER を指定します。
- クライアント認証を必要とするサーバーとして SSL ハンドシェークを実行する場合、GSK_AS_SERVER_WITH_CLIENT_AUTH を指定します。
- 認証なしのクライアントとして SSL ハンドシェークを実行する場合、GSK_AS_CLIENT_NO_AUTH を指定します。

DName フィールドは、キー・データベース・ファイル内の使用したい項目 (証明書) の識別名またはラベルを表す文字ストリングを指定します。NULL を指定すると、デフォルトのキー・データベース・ファイル項目を使用することができます。

sec_type フィールドは、使用されるセキュリティー・プロトコルを指定するヌル終了文字ストリングを指定します。

cipher_specs フィールドは、現在 VSE 下では使用されません。

v3cipher_specs は、使用優先順に並べられた SSL バージョン 3.0 暗号のリストが入っているヌル終了文字ストリングを指定します。システムにインストールされた暗号化のレベルによっては、一部の値は無効である場合があります。有効な値を任意の順序で組み合わせて使用できます。システムでサポートされる暗号仕様の判別については、 157 ページの『gsk_get_cipher_info() — 暗号関連情報の照会』を参照してください。cipher_specs に NULL 値を指定すると、デフォルトの SSL バージョン 3.0 暗号仕様が使用されます。

skread フィールドは、SSL 用に読み取り機能を実行する、アプリケーションによって提供される入出力ルーチンの、エントリー・ポイントを指定します。このアプリケーションは、`fetchep()` を使用して、この入出力ルーチンまたは呼び出されるサブルーチンが書き込み可能な静的変数またはグローバル変数を参照する場合、この入出力ルーチンのエントリー・ポイントを登録する必要があります。このルーチン

のパラメーターは、*skread* に指定されているように定義されなければなりません。SSL は、gsk_secure_soc_init() 呼び出しおよび gsk_secure_soc_read() 呼び出し中に、SSL ハンドシェークを実行する際に *skread* ルーチンを使用します。*skread* ルーチンは、次のようにインプリメントできます。

```
int skread(int fd, void *data, int len){
 return(recv(fd, data, len, 0));
}
```

skwrite フィールドは、SSL 用に書き込み機能を実行する、アプリケーションによって提供される入出力ルーチンの、エントリー・ポイントを指定します。このアプリケーションは、fetchep() を使用して、この入出力ルーチンまたは呼び出されるサブルーチンが書き込み可能な静的変数またはグローバル変数を参照する場合、この入出力ルーチンのエントリー・ポイントを登録する必要があります。このルーチンのパラメーターは、*skwrite* に指定されているように定義されなければなりません。SSL は、gsk_secure_soc_init() 呼び出しおよび gsk_secure_soc_write() 呼び出し中に、SSL ハンドシェークを実行する際に *skwrite* ルーチンを使用します。*skwrite* ルーチンは、次のようにインプリメントできます。

```
int skwrite(int fd, void *data, int len){
 return(send(fd, data, len, 0));
}
```

cipherSelected フィールドは、現在 VSE 下では使用されません。

v3cipherSelected フィールドは、このセッション用に選択された SSL バージョン 3.0 暗号仕様に対応する値を指定します。

failureReasonCode フィールドは、gsk_secure_soc_init() の失敗の理由コードを指定します。

cert_info フィールドは、クライアントの証明書からの識別名コンポーネントを指定します。このパラメーターが有効なのは、SSL を使用するサーバーに対してクライアント認証が要求されている場合だけです。*gsk_cert_info* 構造体は、*gskssl.h* ヘッダー・ファイル内に次のように定義されています。

```
typedef struct _gsk_cert_info { /* Client certificate information    */
   char * cert_body;               /* Certificate body              */
   int    cert_body_len;           /* Lenth of certificate body     */
   char * sessionID;               /* Current session ID            */
   int    newSessionID;            /* TRUE if sid is new            */
   char * serial_num;              /* Serial number                 */
   char * common_name;             /* Common name of client         */
   char * locality;                /* Locality                      */
   char * state_or_province;       /* State or Province             */
   char * country;                 /* Country                       */
   char * org;                     /* Organization                  */
   char * org_unit;                /* Organizational Unit           */
   char * issuer_common_name;      /* Issuer's common name          */
   char * issuer_locality;         /* Issuer's locality             */
   char * issuer_state_or_province; /* Issuer's state or province   */
   char * issuer_country;          /* Issuer's country              */
   char * issuer_org;              /* Issuer's organization         */
   char * issuer_org_unit;         /* Issuer's organizational unit  */
} gsk_cert_info;
```

gsk_data フィールドは、*gsk_init_data* 構造体ポインターを指定します。このフィールドは、gsk_initialize() 関数呼び出し中に使用されたのと同じ *gsk_init_data* 構造体を指すべきです。

戻り値

正常に完了した場合、gsk_secure_soc_init() は、タイプ *gsk_soc_data* の構造体へのポインターを戻します。この構造体は後続の SSL 操作で使用されるので、このポインターを保管してください。*gsk_soc_data* 構造体は、*gsksl.h* ヘッダー・ファイル内に次のように定義されています。

```
typedef struct _gsk_soc_data {
   void * sk_SSLHandle;          /* gskssl connector to SSLHandlestr */
} gsk_soc_data;
```

エラーが発生した場合は、*gsk_soc_init_data* 構造体の *failureReasonCode* フィールドを使用してエラーが示されます。

failureReasonCode フィールドが GSK_ERROR_IO に設定された場合、一般入出力エラーが発生したことを意味し、errno の値によって具体的なエラーが示されます。

注: この操作中に errno が変わることがあります。しかし、SSL インターフェースが明示的に errno を使用することはなく、エラー原因を判別するために errno が使用されることもありません。*gsk_soc_init_data* 構造体の *failureReasonCode* フィールドが、SSL API からのエラーを示す唯一の指標です。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の SSL 関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 NULL を返し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gsk_secure_soc_read() — セキュア・ソケット接続でのデータ受信

フォーマット

```
#include <gskssl.h>

int gsk_secure_soc_read(gsk_soc_data *user_socket,
                        void *data_buffer,
                        int buffer_length);
```

概要

gsk_secure_soc_read() 関数は、アプリケーションが指定する読み取りルーチンを使用して、セキュア・ソケット接続でデータを受信します。

パラメーター
: 説明

user_socket
: データ読み取りに使用されるセキュア・ソケット接続を初期設定した gsk_secure_soc_init() から戻された、*gsk_soc_data* へのポインター。

data_buffer
: ユーザーが用意する、データが保管されるバッファーへのポインター。

buffer_length
: 読み取られるバイト数。これは、*data_buffer* の長さ以下でなければなりません。

戻されるデータの最大長は 32KB を超えることはありません。なぜなら、SSL はレコード・レベル・プロトコルであり、許容される最大レコードは、32KB から、SSL レコード・ヘッダーに必要な長さを引いたものだからです。

gsk_secure_soc_read() の呼び出しに、ソケット読み取り関数 (recv()、read()、readv()、...) のいずれかを不用意に組み合わせることは、可能ではありますが、お勧めしません。これには、クライアント・プログラムとサーバー・プログラムの操作を注意深く突き合わせる必要があります。SSL レコードのどこかの部分がソケット読み取り関数を使用して読み取られた場合、次に gsk_secure_soc_read() が実行されたときに、致命的な SSL プロトコル・エラーが検出されます。

SSL およびソケットの読み取りと書き込みは混合できますが、対応するよう組み合わせて実行しなければなりません。クライアント・アプリケーションが 1 回以上のソケット send() 呼び出しを使用して 100 バイトのデータを書き込む場合、サーバー・アプリケーションは 1 回以上のソケット recv() 呼び出しを使用して正確に 100 バイトのデータを読み取らなければなりません。これは、gsk_secure_soc_read() と gsk_secure_soc_write() にも当てはまります。

SSL はレコード単位のプロトコルなので、SSL がレコード全体を受信してからでないとデータを復号できず、アプリケーションにデータは戻されません。従って、読み取りに対して使用可能なデータがあることを select() が示していても、後続の gsk_secure_soc_read() が、SSL レコードの残りの部分の受信を待機してハングする場合があります。

戻り値

gsk_secure_soc_read() 呼び出しは整数を戻します。0 またはそれより大きい値は、読み取られたバイト数を示しています。0 より小さい値は、エラーが発生したことを示します。

GSK_ERROR_IO が戻された場合、一般入出力エラーが発生したことを意味し、errno の値によって具体的なエラーが示されます。

注: この操作中に errno が変わることがあります。しかし、SSL インターフェースが明示的に errno を使用することはなく、エラー原因を判別するために errno が使用されることもありません。戻り値が、SSL API からのエラーを示す唯一の指標です。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の SSL 関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 GSK_ERROR_UNSUPPORTED を返し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gsk_secure_soc_reset() — セキュリティー・パラメーターのリフレッシュ

フォーマット

```
#include <gskssl.h>

int gsk_secure_soc_reset(gsk_soc_data *user_socket);
```

概要

gsk_secure_soc_reset() 関数は、このセッションのセキュリティー・パラメーター (暗号鍵など) をリフレッシュします。

gsk_secure_soc_reset() は、クライアントまたはサーバーが SSL 環境をリセットする必要があるときに使用します。gsk_secure_soc_reset() は、gsk_secure_soc_init() を正常に呼び出した後にのみ、呼び出してください。また、gsk_secure_soc_reset() は、キャッシュに入れられた SSL セッションの接続を再開または再始動する場合、および、その接続に使用されたキーをリセットする場合にも使用します。

パラメーター
　説明

user_socket
　gsk_secure_soc_init() から戻された *gsk_soc_data* 構造体へのポインター。

戻り値

gsk_secure_soc_reset() 呼び出しは整数を戻します。値 0 は成功を表します。0 より小さい値は、エラーが発生したことを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の SSL 関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 GSK_ERROR_UNSUPPORTED を返し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

関連情報

- 162 ページの『gsk_secure_soc_init() — セキュア・ソケット接続用データ域の初期設定』
- 詳しくは、「*TCP/IP for VSE Optional Features*」を参照してください。

gsk_secure_soc_write() — セキュア・ソケット接続でのデータ送信

フォーマット

```
#include <gskssl.h>

int gsk_secure_soc_write(gsk_soc_data *user_socket,
                         void *data_buffer,
                         int buffer_length);
```

概要

gsk_secure_soc_write() 関数は、アプリケーションが指定する書き込みルーチンを使用して、セキュア・ソケット接続でデータを送信します。

パラメーター
　説明

user_socket
データ書き込みに使用されるセキュア・ソケット接続を初期設定した gsk_secure_soc_init() から戻された、*gsk_soc_data* へのポインター。

data_buffer
ユーザーが用意する、書き込まれるデータが保管されるバッファーへのポインター。

buffer_length
書き込まれるバイト数。これは、*data_buffer* の長さ以下でなければなりません。

注: 現在 SSL for VSE は、一度の gsk_secure_soc_write() 呼び出しで最大 64KB までの送信をサポートしています。

SSL アプリケーションに送信されるアプリケーション・データが 32KB より大きい場合、アプリケーション・データ全体を読み取るために、複数回の gsk_secure_soc_read() 呼び出しを実行する必要があります。

SSL およびソケットの読み取りと書き込みは混合できますが、対応するよう組み合わせて実行しなければなりません。クライアント・アプリケーションがソケット送信を 1 回以上呼び出して 100 バイトのデータを書き込む場合、サーバー・アプリケーションはソケット受信を 1 回以上呼び出して正確に 100 バイトのデータを読み取らなければなりません。これは、gsk_secure_soc_read() と gsk_secure_soc_write() にも当てはまります。書き込みバッファーが複数のバッファーに分割されている場合、セキュア・ソケット接続のリモート・サイトは、そのバッファー全体を読み取るために十分な回数の gsk_secure_soc_read() 操作を実行する必要があります。

戻り値

gsk_secure_soc_write() 呼び出しは整数を戻します。0 またはそれより大きい値は、書き込まれたバイト数を示しています。0 より小さい値は、エラーが発生したことを示します。

GSK_ERROR_IO が戻された場合、一般入出力エラーが発生したことを意味し、errno の値によって具体的なエラーが示されます。

注: この操作中に errno が変わることがあります。しかし、SSL インターフェースが明示的に errno を使用することはなく、エラー原因を判別するために errno が使用されることもありません。戻り値が、SSL API からのエラーを示す唯一の指標です。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の SSL 関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 GSK_ERROR_UNSUPPORTED を返し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gsk_uninitialize() — 現行の SSL/TLS 環境設定の除去

フォーマット

```
#include <gskssl.h>

int gsk_uninitialize(void);
```

概要

gsk_uninitialize() 関数は、SSL 環境に関する現行の設定全体を除去します。gsk_uninitialize() は、セッションのタイムアウト値や SSL プロトコルなどの設定値を除去します。

gsk_uninitialize() は、SSL 環境設定をリセットする必要がある場合に使用します。その後、gsk_initialize() を使用して、SSL 環境設定の新しい集合を作成します。

注: gsk_uninitialize() を呼び出す前に、現行の SSL 環境を使用して作成された SSL セッションはすべてクローズされていなければなりません。

戻り値

gsk_uninitialize() 呼び出しは整数を戻します。値 0 は成功を表します。0 より小さい値は、エラーが発生したことを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の SSL 関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 GSK_ERROR_UNSUPPORTED を返し、 は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

gsk_user_set() — コールバック・ルーチンの提供

フォーマット

```
#include <gskssl.h>

int gsk_user_set(int user_data_fid,
                 void *user_input_data,
                 void *reserved);
```

概要

gsk_user_set() 関数を使用すると、SSL アプリケーションは、デフォルトの SSL インプリメンテーションを使用するのではなく、コールバックを提供することができます。

注: gsk_user_set() 関数は、現在 VSE 下では使用されません。

パラメーター
: 説明

user_data_fid
: 実行するアクションを指定する整数値。

user_input_data
: アクション固有の情報を指定するポインター。

予約済み
: 将来の SSL での使用のため予約済みであり、ヌルに設定されなければなりません。

戻り値

gsk_user_set() 呼び出しは整数を戻します。値 0 は成功を表します。0 より小さい値は、エラーが発生したことを示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の SSL 関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 GSK_ERROR_UNSUPPORTED を返し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

htonl() — ホストからネットワークへのアドレス長整数の変換

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1

#include <inet.h>
in_addr_t htonl (in_addr_t hostlong);
```

概要

htonl() 呼び出しは、長整数をホスト・バイト・オーダーからネットワーク・バイト・オーダーに変換します。

パラメーター
: 説明

hostlong
: ネットワーク・バイト・オーダーに変換される符号なし長整数。

注: IBM Z の場合、ホスト・バイト・オーダーとネットワーク・バイト・オーダーは同じです。ただし、プラットフォーム間の移植性のため、ホスト・バイト・オーダーとネットワーク・バイト・オーダー間の変換が必要な場合はこのルーチンを使用することをお勧めします。

戻り値

htonl()は、変換された長整数を戻します。

htons() — ネットワーク・バイト・オーダーへの符号なし短整数の変換

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <inet.h>

in_port_t htons(in_port_t hostshort);
```

概要

htons() 呼び出しは、短整数をホスト・バイト・オーダーからネットワーク・バイト・オーダーに変換します。

パラメーター
　　説明

hostshort
　　ネットワーク・バイト・オーダーに変換される符号なし短整数。

注: IBM Z の場合、ホスト・バイト・オーダーとネットワーク・バイト・オーダーは同じです。ただし、プラットフォーム間の移植性のため、ホスト・バイト・オーダーとネットワーク・バイト・オーダー間の変換が必要な場合はこのルーチンを使用することをお勧めします。

戻り値

htons() は、変換された短整数を戻します。

if_freenameindex() — if_nameindex() によるメモリー割り振りの解放

フォーマット

```
#define _OPEN_SYS_SOCK_IPV6
#include <net/if.h>


void if_freenameindex(struct if_nameindex *ptr);
```

概要

if_freenameindex() 関数は、if_nameindex() によって割り振られたメモリーを解放します。*ptr* 引数は、if_nameindex() によって戻されたポインターである必要があります。

戻り値

戻り値は定義されません。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

if_indextoname() - ネットワーク・インターフェース索引の対応する名前へのマッピング

フォーマット

```
#define _OPEN_SYS_SOCK_IPV6
#include <net/if.h>


char *if_indextoname(unsigned int ifindex, char *ifname);
```

概要

if_indextoname() 関数は、インターフェース索引を、対応するインターフェース名にマップします。この関数が呼び出される場合、*ifname* は少なくとも IF_NAMESIZE バイトのバッファーを示す必要があります。このバッファーに、イ

ンターフェース索引 *ifindex* に対応したインターフェース名が戻されます。これが正常に実行されなかった場合、関数は NULL ポインターを戻し、errno を設定してエラーを示します。

戻り値

エラー・コード
: 説明

EINVAL
: ifindex パラメーターがゼロか、または *ifname* パラメーターが NULL、あるいはその両方でした。

ENOMEM
: インターフェース名の情報を取得するには、使用可能なストレージが不十分です。

ENXIO
: *ifindex* がインターフェース名を作成しません。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

if_nameindex() - すべてのネットワーク・インターフェース名と索引を返す

フォーマット

```
#define _OPEN_SYS_SOCK_IPV6
#include <net/if.h>


struct if_nameindex *if_nameindex(void);
```

概要

if_nameindex() 関数は、if_nameindex 構造体の配列を戻します。インターフェース 1 つにつき、構造体 1 つです。配列の終了は、if_index がゼロ、および if_name が NULL の構造体で示されます。if_nameindex 構造体は、単一インターフェースの情報を保持し、<net/if.h> ヘッダーをインクルードした結果として定義されます。

```
struct if_nameindex {
   unsigned int if_index; /* 1, 2, ... */
   char *if_name; /* null terminated name: "le0", ... */
};
```

この構造体の配列に使用されたメモリーと、if_name メンバーによって指されたインターフェース名は、動的に取得されます。このメモリーは、if_freenameindex() 関数を呼び出すことによって解放されます。

戻り値

正常に実行された場合、if_nameindex() は if_nameindex 構造体の配列を指すポインターを戻します。正常に実行されなかった場合、if_nameindex() は NULL を戻し、errno を次のいずれかに設定します。

エラー・コード
説明

ENOMEM
配列の提供には、使用可能なストレージが不十分です。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

if_nametoindex() - ネットワーク・インターフェース名の対応する索引へのマッピング

フォーマット

```
#define _OPEN_SYS_SOCK_IPV6
#include <net/if.h>


unsigned int if_nametoindex(const char *ifname);
```

概要

if_nametoindex() 関数は、インターフェース名 *ifname* に対応したインターフェース索引を戻します。

戻り値

正常に実行された場合は、if_nametoindex() はインターフェース名 *ifname* に対応するインターフェース索引を戻します。正常に実行されなかった場合、if_nametoindex() は 0 を戻して errno を次のいずれかに設定します。

エラー・コード
説明

EINVAL
無効なパラメーターが指定されました。*ifname* パラメーターは NULL です。

ENOMEM
インターフェース名の情報を取得するには、使用可能なストレージが不十分です。

ENXIO
ifname パラメーターで提供された、指定インターフェース名は存在しません。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

inet_addr() — ネットワーク・バイト・オーダーへの IP アドレスの変換

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <inet.h>
#include <in.h>

in_addr_t inet_addr(const char *cp);
```

概要

inet_addr() 呼び出しは、標準のドット 10 進表記で表されたホスト・アドレスの文字ストリングを解釈し、IP アドレスとして使用できるホスト・アドレスを戻します。

パラメーター

	説明
cp	標準のドット 10 進 (.) 表記の文字ストリング

標準のドット 10 進表記で指定された値は、以下のいずれかの形式です。

```
a.b.c.d
a.b.c
a.b
a
```

4 つのパートで構成されるアドレスが指定された場合、各パートが 1 バイトのデータと解釈され、左から右へ、IP アドレスの 4 バイトの 1 つずつに割り当てられます。

3 つのパートで構成されるアドレスが指定された場合、最後のパートは 16 ビットと解釈され、ネットワーク・アドレスの右端の 2 バイトになります。3 パートのアドレス・フォーマットは、**128.net.host** のようにクラス B ネットワーク・アドレスを指定する場合に便利です。

2 つのパートで構成されるアドレスが指定された場合、最後のパートは 24 ビットと解釈され、ネットワーク・アドレスの右端の 3 バイトになります。2 パートのアドレス・フォーマットは、**net.host** のようにクラス A ネットワーク・アドレスを指定する場合に便利です。

1 つのパートで構成されるアドレスが指定された場合、バイトの再構成を行うことなく、その値が直接ネットワーク・アドレス・スペースに保管されます。

標準のドット 10 進表記でアドレス部として与えられる数は、10 進数、16 進数、または 8 進数です。数は C 言語構文で解釈されます。0x で始まる場合は 16 進数、0 で始まる場合は 8 進数、0 で始まらない数字は、10 進数を意味します。

戻り値

IP アドレスがネットワーク・バイト・オーダーで戻されます。戻された IP アドレスにエラーがある (例えば、形式が正しくない) 場合、戻り値は INADDR_NONE (-1) になります。INADDR_NONE は、**in.h** インクルード・ファイルに定義されています。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 INADDR_NONE (-1) を戻します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 INADDR_NONE (-1) を戻します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

inet_lnaof() — ホスト・バイト・オーダーへのローカル・ネットワーク・アドレスの変換

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <inet.h>

in_addr_t inet_lnaof(struct in_addr in);
```

概要

inet_lnaof() 呼び出しは、インターネット・ホスト・アドレスを分解し、ローカル・ネットワーク・アドレス部分を戻します。

パラメーター

	説明
in	ホスト IP アドレス

戻り値

ローカル・ネットワーク・アドレスが、ホスト・バイト・オーダーで戻されます。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

inet_makeaddr() — インターネット・ホスト・アドレスの作成

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <inet.h>

struct in_addr inet_makeaddr(in_addr_t net, in_addr_t lna);
```

概要

inet_makeaddr() 呼び出しは、指定されたネットワーク番号とローカル・ネットワーク・アドレスから IP アドレスを組み立てます。

パラメーター

	説明
net	ネットワーク番号
lna	ローカル・ネットワーク・アドレス

戻り値

IP アドレスがネットワーク・バイト・オーダーで戻されます。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に in_addr struct を返し、フィールド s_addr は -1 に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

inet_netof() — インターネット・ホスト・アドレスからのネットワーク番号の取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <inet.h>

in_addr_t inet_netof(struct in_addr in);
```

概要

inet_netof() 呼び出しは、インターネット・ホスト・アドレスを分解し、ネットワーク番号部分を戻します。

パラメーター

	説明
in	ネットワーク・バイト・オーダーの IP アドレス

戻り値

ネットワーク番号がホスト・バイト・オーダーで戻されます。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

inet_network() — 10 進ホスト・アドレスからのネットワーク番号の取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <inet.h>

in_addr_t inet_network(const char *cp);
```

概要

inet_network() 呼び出しは、標準のドット 10 進表記で表されたアドレスの文字ストリングを解釈し、ネットワーク番号として使用できる数値を戻します。

パラメーター

	説明
cp	標準のドット 10 進 (.) 表記の文字ストリング

戻り値

ネットワーク番号がホスト・バイト・オーダーで戻されます。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

inet_ntoa() — 10 進インターネット・ホスト・アドレスの取得

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <inet.h>

char *inet_ntoa(struct in_addr in);
```

概要

inet_ntoa() 呼び出しは、ドット 10 進表記で表されたストリングへのポインターを戻します。inet_ntoa() は、ネットワーク・バイト・オーダーで 32 ビットとして表現された IP アドレスを受け入れ、ドット 10 進表記で表されたストリングを戻します。

パラメーター

	説明
in	ホスト IP アドレス

戻り値

ドット 10 進表記で表された IP アドレスへのポインターが戻されます。ポインターが指すストレージは個別のスレッド上に存在し、後続の呼び出しによって上書きされます。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に NULL ポインターを返します。この場合、メッセージ EDCV001I または EDCT002I が出されます。

inet_ntop() - IP アドレス形式の 2 進数からテキストへの変換

フォーマット

```
#define _OPEN_SYS_SOCK_IPV6
#include <inet.h>


const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);
```

概要

inet_ntop() 関数は、*src* で指定された、バイナリー形式の IP アドレスを、標準テキスト形式に変換し、その結果を *dst* に入れます。ただし、*dst* 内の使用可能スペース *size* が十分にあることが前提になります。引数 *af* は、IP アドレスのファミリーを指定します。AF_INET または AF_INET6 を指定できます。

引数 *src* は、*af* 引数が AF_INET の場合は、IPv4 インターネット・アドレス、*af* 引数が AF_INET6 の場合は、IPv6 インターネット・アドレスが入ったバッファーを指します。このアドレスは、ネットワーク・バイト・オーダーでなければなりません。

引数 *dst* は、この関数が結果のテキスト・ストリングを保管するバッファーを指します。*size* 引数は、このバッファーのサイズを指定します。アプリケーションは、非 NULL の *dst* 引数を指定する必要があります。IPv6 アドレスの場合、バッファーは 46 バイト以上が必要です。IPv4 アドレスの場合、バッファーは 16 バイト以上が必要です。

IPv4 および IPv6 アドレスをストリング形式で保管するための適切なサイズのバッファーをアプリケーションで簡単に宣言できるようにするため、次の 2 つの定数が **in.h** に定義されています。

```
#define INET_ADDRSTRLEN 16
#define INET6_ADDRSTRLEN 46
```

注**:** inet_ntop() 関数は、拡張 ASCII の拡張機能のレベルに依存します。

戻り値

正常に実行された場合、inet_ntop() は、変換されたアドレスを含むバッファーへのポインターを戻します。正常に実行されなかった場合、inet_ntop() は NULL を戻して、errno を次のいずれかの値に設定します。

エラー・コード
: 説明

EAFNOSUPPORT
: *af* に指定されたアドレス・ファミリーは、サポートされません。

ENOSPC
: 宛先バッファー *size* が、小さすぎます。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

inet_pton() - IP アドレス形式のテキストから 2 進数への変換

フォーマット

```
#define _OPEN_SYS_SOCK_IPV6
#include <inet.h>


int inet_pton(int af, const char *src, void *dst);
```

概要

inet_pton() 関数は、標準テキスト・フォーマットの IP アドレスを、数値バイナリー・フォーマットに変換します。引数 *af* は、アドレスのファミリーを指定します。

注**:** AF_INET および AF_INET6 アドレス・ファミリーは、現在サポートされています。

引数 *src* は、渡されるストリングを指します。引数 *dst* は、inet_pton() が、数値アドレスを保管するバッファーを指します。アドレスは、ネットワーク・バイト順で戻されます。呼び出し元は、*dst* が指すバッファーが十分に大きく、数値アドレスを保持できるか確認する必要があります。

af 引数が AF_INET の場合、inet_pton() は、標準 IPv4 小数点付き 10 進数表記のストリングを受け入れます。

```
ddd.ddd.ddd.ddd
```

ここで、*ddd* は、0 から 255 までの 1 から 3 桁の 10 進数です。

af 引数が AF_INET6 の場合、*src* ストリングは、以下の標準 IPv6 テキスト形式の中のいずれかの形式である必要があります。

1. 優先される形式は *x:x:x:x:x: x:x: x:x:* です。*x* は、アドレスの 8 つの 16 ビットの 16 進数値です。個々のフィールドの先行ゼロは省略できますが、各フィールドに少なくとも 1 つの数表示がある必要があります。
2. 優先される形式での、連続するゼロ・フィールドのストリングは、*::* と示されます。*::* は、1 つのアドレスの中では、一度だけ表示されます。未指定のアドレス *(0:0:0:0:0:0:0:0:)* は、単に *::* と表現されます。
3. IPv4 と IPv6 の混合環境を扱う際に、より使いやすい 3 つ目の形式として、*x:x:x:x:x:x:d.d.d.d.* を使用できます。この場合の *x* はアドレスの 6 つの高位 16 ビットの 16 進値であり、*d* は、アドレスの 4 つの下位 8 ビットの 10 進数値です (標準 IPv4 表記)。

注**:** inet_pton() 関数は、拡張 ASCII の拡張機能のレベルに依存します。

戻り値

正常に終了した場合、inet_pton() は 1 を戻し、バイナリー・フォーマットの IP アドレスを、dst が指すバッファーに保管します。

src が指す入力バッファーが有効なストリングでないために、正常に終了しなかった場合、inet_pton() は 0 を戻します。

af 引数が知られていないという理由で正常に実行されなかった場合、inet_pton() は -1 を戻して、errno を次のいずれかの値に設定します。

エラー・コード
: 説明

EAFNOSUPPORT
: *af* に指定されたアドレス・ファミリーは、サポートされません。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

initapi() — サブタスク用ソケット API の初期化

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

int initapi(int maxsock, const char *taskid);
```

概要

initapi() 関数は、ソケット API を初期化し、VSE サブタスク用のソケットの最大数とサブタスクの ID を設定します。

パラメーター
　説明

maxsock
　サブタスクで使用されるソケットの最大数。

taskid
　サブタスクの ID として設定する名前。

戻り値

関数 initapi は、アプリケーションに割り当て可能な記述子の最大値を返します。正の値は成功を表し、値 -1 はエラーを表します。具体的なエラーを errno の値が示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

ioctl() — ソケットの制御

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <ioctl.h>
int ioctl(int socket, int cmd, ... /* arg */);
```

概要

ioctl() は、ソケットに対してさまざまな制御機能を実行します。

cmd 引数は、対象となるソケットに依存し、実行する制御機能を選択します。

arg 引数は、要求された機能を実行するためにこの特定のデバイスが必要とする追加情報を表します。*arg* のタイプは、特定の制御要求に依存しますが、整数、またはデバイス特有のデータ構造体へのポインターのいずれかです。

ソケット

ソケットでは以下の ioctl() コマンドが使用されます。

コマンド
　説明

FIONBIO
ソケットに対する非ブロッキング I/O を設定またはクリアします。*arg* は整数へのポインターです。整数が 0の場合、ソケット上の非ブロッキング I/O がクリアされます。それ以外の場合、ソケットは非ブロッキング I/O 用に設定されます。

SIOCGHOMEIF6
IPv6 ホーム・インターフェースを取得します。*arg* は、**ioctl.h** で定義されている **NetConfHdr** 構造体へのポインターです。ホーム・インターフェースのリストを含む **HomeIf** 構造体へのポインターが、引数で指示された **NetConfHdr** 内に返されます。

SIOCGIFADDR
ネットワーク・インターフェース・アドレスを取得します。*arg* は、**if.h** で定義されている **ifreq** 構造体へのポインターです。インターフェース・アドレスが引数内に返されます。このオプションは、AF_INET ドメインの場合にのみ有効です。このマクロは、_OPEN_SYS_IF_EXT フィーチャーによって保護されます。

SIOCGIFBRDADDR
ネットワーク・インターフェース・ブロードキャスト・アドレスを取得します。*arg* は、**if.h** で定義されている **ifreq** 構造体へのポインターです。インターフェース・ブロードキャスト・アドレスが引数内に返されます。このオプションは、AF_INET ドメインの場合にのみ有効です。このマクロは、_OPEN_SYS_IF_EXT フィーチャーによって保護されます。

SIOCGIFCONF
ネットワーク・インターフェース構成を取得します。*arg* は、**if.h** で定義されている **ifconf** 構造体へのポインターです。**ifconf** 構造体で指示されたバッファー内にインターフェース構成が返されます。戻されたデータの長さが、前にバッファーの長さが含まれていたフィールドに戻されます。このオプションは、AF_INET ドメインの場合にのみ有効です。このマクロは、_OPEN_SYS_IF_EXT フィーチャーによって保護されます。

SIOCGIFDSTADDR
ネットワーク・インターフェース宛先アドレスを取得します。*arg* は、**if.h** で定義されている **ifreq** 構造体へのポインターです。インターフェース宛先 (Point-to-Point) アドレスが引数内に返されます。このオプションは、AF_INET ドメインの場合にのみ有効です。このマクロは、_OPEN_SYS_IF_EXT フィーチャーによって保護されます。

端末とソケットの戻り値

値 0 は成功を表します。値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
説明

EBADF
socket パラメーターが無効ソケット記述子です。

EINVAL
要求が無効か、またはサポートされていません。

EMVSPARM
無効なパラメーターがサービスに渡されました。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

例

ioctl() 呼び出しの例を以下に示します。

```
int s;
int dontblock;
int rc;
.
.
.
/* Place the socket into nonblocking mode */
dontblock = 1;
rc = ioctl(s, FIONBIO, (char *) &dontblock);
.
.
.
```

listen() — 着信クライアント要求のためのサーバーの準備

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

int listen(int socket, int backlog);
```

概要

listen() 呼び出しは、ストリーム・ソケットのみに適用できます。この関数は、クライアント接続要求を受け入れる準備を完了し、着信接続要求を待ち行列に入れるための、*backlog* の長さの接続要求待ち行列を作成します。いったん一杯になると、以降の接続要求はリジェクトされます。

パラメーター
説明

socket ソケット記述子。

backlog
保留接続の待ち行列の最大長を定義します。このパラメーターは無視されます。常に値 1 が想定されます。

listen() 呼び出しは、クライアント接続要求を受け入れる準備ができていることを示します。これは、能動ソケットを受動ソケットに変換します。いったん呼び出されると、接続要求を開始するために、*socket* を能動ソケットとして使用することはできなくなります。listen() の呼び出しは、サーバーが接続を受け入れる 4 つのステップのうちの 3 つ目のステップです。socket() でストリーム・ソケットを割り振り、bind() で *socket* に名前を結合した後に、この関数が呼び出されます。これは accept() を呼び出す前に呼び出されなければなりません。

backlog が 0 より小さい場合、*backlog* は 0 に設定されます。backlog が SOMAXCONN (**socket.h** で定義された) よりも大きい場合、*backlog* は SOMAXCONN に設定されます。

その値はインストール済みの TCP/IP で可能な接続の最大数を超えることはできません。

戻り値

値 0 は成功を表します。値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
: 説明

EBADF
: *socket* パラメーターが無効ソケット記述子です。

EDESTADDRREQ
: ソケットがローカル・アドレスにバインドされていません。またプロトコルがバインドされていないソケットを listen することをサポートしていません。

EINVAL
: 無効な引数が指定されました。ソケットに名前が付いていない (bind() が行われていない) か、またはソケットが接続の受け入れ可能状態です (listen() が既に行われている)。ソケットはすでに接続されています。

ENOBUFS
: 使用可能なシステム・リソースが不十分で、呼び出しを完了させることができません。

EOPNOTSUPP
: *socket* パラメーターが、listen() 呼び出しをサポートするソケット記述子ではありません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

maxdesc() - デフォルト範囲を超えて拡張するためのソケット番号の取得

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <types.h>
#include <socket.h>


int maxdesc(int *totdesc, int *inetdesc);
```

概要

バルク・モード・ソケットはサポートされていません。この関数を使用しないでください。

戻り値

正常に実行された場合、maxdesc() は 0 を戻します。

正常に実行されなかった場合、maxdesc() は -1 を戻して、errno を次のいずれかの値に設定します。

エラー・コード
: 説明

EALREADY
: ソケットの作成後、setibmsockopt() の呼び出し後、または直前の maxdesc() の呼び出し後に、プログラムで maxdesc() を呼び出しました。

EFAULT
: 指定された totdesc パラメーターの使用により、呼び出し元のアドレス・スペース外のストレージ、または呼び出し元によって更新できないストレージにアクセスしようとしました。

ENOMEM
: ユーザーのアドレス・スペースのストレージが不十分です。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

ntohl() — ホスト・バイト・オーダーへの長整数の変換

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <inet.h>

in_addr_t ntohl(in_addr_t netlong);
```

概要

ntohl() 呼び出しは、長整数をネットワーク・バイト・オーダーからホスト・バイト・オーダーへ変換します。

パラメーター
: 説明

netlong
: ホスト・バイト・オーダーに変換される符号なし長整数。

注: IBM Z の場合、ホスト・バイト・オーダーとネットワーク・バイト・オーダーは同じです。ただし、プラットフォーム間の移植性のため、ホスト・バイト・オーダーとネットワーク・バイト・オーダー間の変換が必要な場合はこのルーチンを使用することをお勧めします。

戻り値

ntohl() は、変換後の長整数を戻します。

ntohs() — ホスト・バイト・オーダーへの符号なし短整数の変換

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <inet.h>

in_port_t ntohs(in_port_t netshort);
```

概要

ntohs() 呼び出しは、短整数をネットワーク・バイト・オーダーからホスト・バイト・オーダーへ変換します。

パラメーター
　説明

netshort
　ホスト・バイト・オーダーに変換される符号なし短整数。

注: IBM Z の場合、ホスト・バイト・オーダーとネットワーク・バイト・オーダーは同じです。ただし、プラットフォーム間の移植性のため、ホスト・バイト・オーダーとネットワーク・バイト・オーダー間の変換が必要な場合はこのルーチンを使用することをお勧めします。

戻り値

ntohs() は、変換後の短整数を戻します。

poll() — ソケット記述子でのアクティビティーのモニター

形式 1

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <poll.h>

int poll(struct pollfd listptr[], nfds_t nmsgsfds, int timeout);
```

形式 2

```
#define _XOPEN_SOURCE_EXTENDED 1
#define _OPEN_MSGQ_EXT
#include <types.h>
#include <time.h>
#include <poll.h>

int poll(void *listptr, nmsgsfds_t nmsgsfds, int timeout);
```

概要

poll() 関数は、ソケット記述子で入出力を多重化するためのメカニズムをアプリケーションに提供します。

poll() は、*listptr* によって指示された配列のメンバーごとに、メンバー内で指定されたイベントがないか、指定のソケット記述子を検査します。配列内の *pollfd* 構造体の数は、*nmsgsfds* によって指定されます。poll() 関数は、アプリケーションがデータの読み取りまたは書き込みが可能なソケット記述子、またはエラー・イベントが起こったファイル記述子を識別します。

パラメーター
説明

listptr
pollfd 構造体の配列を指すポインター。各構造体は、1 つのソケット記述子と、このソケットにとって重要なイベントを指定します。ソケット記述子を監視するためには、*nmsgsfds* の高位ハーフワードを 0 に設定し、下位ハーフワードを提供される *pollfd* 構造体数に設定してから、*pollfd* 構造体の配列を指すポインターを渡します。

nmsgsfds
listptr によって指示された *pollfd* 構造体の数。このパラメーターは、2 つの部分に分かれます。最初の半分 (高位 16 ビット) は、メッセージ・キュー ID 用に予約されています。後の半分 (下位 16 ビット) は、チェックするためのソケット記述子を含む *pollfd* 構造体数を指定します。*nmsgsfds* パラメーターのいずれか半分が値 0 に等しい場合には、対応する構造体が存在していないと考えられます。

timeout
イベントの発生を待機する時間 (ミリ秒)。定義されているイベントが選択した記述子で発生しなかった場合、`poll()` は選択した記述子のいずれかでのイベントの発生を少なくとも *timeout* ミリ秒間待機します。*timeout* の値が 0 の場合、`poll()` は即時に戻ります。*timeout* の値が -1 の場合、`poll()` は、要求のイベントが起こるか、あるいはその呼び出しが中断されるまでブロックします。

各 *pollfd* 構造体には、以下のフィールドが含まれています。
fd オープンされているソケット記述子
events
要求されたイベント
revents
返されたイベント

events フィールドと *revents* フィールドは、イベント・フラグの組み合わせを OR することで構成されたビット・マスクです。

次のマクロを使用すると、*nmsgsfds* パラメーターおよび `poll()` からの戻り値を操作することができます。

マクロ 説明

_SET_FDS_MSGS(*nmsgsfds*, *nmsgs*, *nfds***)**
nmsgsfds の高位ハーフワードを *nmsgs* に設定し、*nmsgsfds* の下位ハーフワードを *nfds* に設定します。

_NFDS(*n***)**
`poll()` からの戻り値 *n* が負ではない場合には、読み取り、書き込み、および例外の基準を満たすソケット記述子の数が返されます。1 つの記述子でも、それが複数の指定基準を満たす場合には、複数回カウントされる場合があります。

_NMSGS(*n***)**
`poll()` からの戻り値 *n* が負ではない場合には、読み取り、書き込み、およ

び例外の基準を満たすメッセージ待ち行列の数を戻します。メッセージ・キューが複数の指定基準に該当する場合、そのメッセージ・キューは複数回カウントされる場合があります。

戻り値

正常に実行された場合、poll() は負でない値を返します。正の値は、ソケット記述子で作動可能になっていることが判明したイベントの合計数を示します。戻り値は *nmsgsfds* と同じになっていて、そこでは戻り値の高位 16 ビットがメッセージ・キューに関連した数を指定し、下位 16 ビットがソケット記述子に関連した数を指定します。

値 0 は、呼び出しがタイムアウトになり、ソケット記述子が選択されていないことを示します。失敗した場合、poll() は -1 を返し、エラーを示すために errno を設定します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

read() — ソケットからの読み取り

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <unistd.h>

ssize_t read(int fs, void *buf, ssize_t N   );
```

概要

read() 関数は、ソケット記述子 *fs* で表されるソケットから、*buf* で表されるメモリー域に、入力の *N* バイトを読み込みます。正常に実行された場合、read() は、読み取ったバイト数によってファイル・オフセットを変更します。*N* は、INT_MAX (limits.h ヘッダー・ファイルで定義されている) より大きくてはなりません。

read() は、フラグを設定しない recv() と同等です。

パラメーター
: 説明

fs
: ソケット記述子。

buf
: データを受け取るバッファーへのポインター。

N
: *buf* パラメーターが指すバッファーの長さ (バイト単位)。

ソケットの動作

read() 呼び出しは、記述子 *fs* のソケットでデータを読み取り、そのデータをバッファーに保管します。read() 呼び出しは、接続されたソケットにのみ適用されます。この呼び出しは、最大 *N* バイトまでのデータを戻します。使用可能なバイトが要求したバイトより少ないと、呼び出しは、現在使用可能な数を戻します。ソケット *fs* でデータが使用できず、ソケットがブロッキング・モードの場合、read() 呼び出しはデータが到着するまで呼び出し元をブロックします。データが使用可能で

なく、ソケットが非ブロッキング・モードの場合、read() は -1 を戻し、エラー・コードを EWOULDBLOCK に設定します。非ブロッキング・モードの設定方法の説明については、 180 ページの『ioctl() — ソケットの制御』または 122 ページの『fcntl() — オープンされたソケット記述子の制御』を参照してください。

データグラム・ソケットの場合、この呼び出しは、データグラムが指定のバッファーに収まる場合は、送信されたデータグラム全体を戻します。余分なデータグラム・データは廃棄されます。ストリーム・ソケットは、データを分離する境界のない情報ストリームのように動作します。例えば、アプリケーション A および B がストリーム・ソケットと接続され、アプリケーション A が 1000 バイトを送信した場合には、この関数のそれぞれの呼び出しは、1 バイト、または 10 バイト、あるいは 1000 バイト全体を戻すことができます。したがって、ストリーム・ソケットを使用するアプリケーションは、この呼び出しをループに入れて、すべてのデータを受信するまで、この関数を呼び出す必要があります。

戻り値

正常に実行された場合、read() は、実際に読み取られて *buf* に入れられたバイト数を戻します。この数は *N* 以下です。値 -1 はエラーを表します。値 `0` は、接続がクローズされていることを示します。

read() は、失敗した場合、値 -1 を戻し、errno を以下のいずれかに設定します。

EBADF
fs が無効ソケット記述子です。

ECONNRESET
接続はピアによって強制的にクローズされました。

EFAULT
buf および *N* パラメーターを使用すると、呼び出し元アドレス・スペースの外側のメモリーへのアクセスを試みる結果になります。

EINVAL
N に、0 より小さい値が含まれているか、または要求が無効か、サポートされていないか、あるいは *fs* によって参照された STREAM またはマルチプレクサーが、マルチプレクサーから (直接または間接に) リンクされたダウンストリームです。

EIO 入出力エラーが発生しました。

ENOBUFS
使用可能なシステム・リソースが不十分で、呼び出しを完了させることができません。

ENOTCONN
接続されていないコネクション指向ソケットで受信しようとしました。

ETIMEDOUT
接続の確立中またはアクティブな接続での伝送のタイムアウトのために、接続がタイムアウトになりました。

EWOULDBLOCK
ソケットは非ブロッキング・モードであり、データが読み取りに使用可能でありません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

例

以下に示すのは、read() 呼び出しの例です。

```
#include <stdio.h>

/* Read from the socket aSocket
   and print number of byte read and string read.
   Return number of bytes read or -1 for no success.
 */
int readFromSocket(int aSocket)
{ int numberOfBytesReceived;
  char dataBuffer 255 ;             /* data to read */

  numberOfBytesReceived=
    read(aSocket, dataBuffer, sizeof(dataBuffer));
  if (numberOfBytesReceived < 0)
  { perror("read"); return -1; }
  else
  { dataBuffer numberOfBytesReceived =0;
    printf("Read string '%s' (length %d).¥n",
           dataBuffer,numberOfBytesReceived);
    return numberOfBytesReceived;
  }
}
```

readv() — ソケットのデータの読み取りとバッファー・セットへの保管

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <uio.h>

ssize_t readv(int fs, const struct iovec *iov, int iovcnt);
```

概要

readv() 呼び出しは、記述子 *fs* のソケットからデータを読み取り、それをバッファー・セットに保管します。データは、iov[0]...iov[iovcnt-1] で指定されたバッファーに分散されます。

パラメーター
　説明

fs　ソケット記述子。

iov　**iovec** 構造体に対するポインター。

iovcnt　*iov* パラメーターが指すバッファー数。

iovec 構造体は **uio.h** で定義されており、以下のフィールドが含まれています。

エレメント
　説明

iov_base
バッファーへのポインター。

iov_len
バッファーの長さ。

記述子は、接続されたソケットを参照します。

この呼び出しは、*iov_len* フィールドすべての合計と等しく、それを超えないバイト数のデータを戻します。要求より少ないバイト数しか使用可能でない場合には、この呼び出しは、現在使用可能なバイト数を戻します。ソケット *fs* のデータが使用できず、さらにソケットがブロック・モードになっている場合には、データが到着するまで、readv() 呼び出しは呼び出し元をブロックします。データが使用できず、さらに *fs* が非ブロック・モードである場合には、readv() は -1 を戻し、エラー・コードは、EWOULDBLOCK に設定されます。

戻り値

正常に実行された場合は、バッファーに読み込まれたバイト数が返されます。値 -1 はエラーを表します。具体的なエラーを errno の値が示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

recv() — ソケットでのデータ受信

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

ssize_t recv( int     socket,
              void   *buf,
              size_t  len,
              int     flags);
```

概要

recv() 呼び出しは、記述子 *socket* のソケットでデータを受信し、そのデータをバッファーに保管します。recv() 呼び出しは、接続されたソケットだけに適用されます。

この呼び出しは、着信メッセージまたはデータの長さを戻します。データグラム・パケットが長すぎて、提供されたバッファーに収まらない場合には、データグラム・ソケットは、余分なバイトを廃棄します。ソケット *socket* でデータが使用できず、*socket* がブロッキング・モードの場合、recv() 呼び出しはデータが到着するまで呼び出し元をブロックします。データが使用可能でなく、*socket* が非ブロッキング・モードの場合、recv() は -1 を戻し、エラー・コードを EWOULDBLOCK に設定します。非ブロッキング・モードの設定方法の説明については、122 ページの『fcntl() — オープンされたソケット記述子の制御』または 180 ページの『ioctl() — ソケットの制御』を参照してください。

データグラム・ソケットの場合、この呼び出しは、データグラムが指定のバッファーに収まる場合は、送信されたデータグラム全体を戻します。ストリーム・ソケットは、データを分離する境界のない情報ストリームのように動作します。例えば、アプリケーション A および B がストリーム・ソケットと接続され、アプリケーション A が 1000 バイトを送信した場合には、この関数のそれぞれの呼び出しは、1 バイト、または 10 バイト、あるいは 1000 バイト全体を戻すことができます。したがって、ストリーム・ソケットを使用するアプリケーションは、この呼び出しをループに入れて、すべてのデータを受信するまで、この関数を呼び出す必要があります。

パラメーター
: 説明

ソケット
: ソケット記述子。

buf
: データを受け取るバッファーへのポインター。

len
: *buf* パラメーターが指すバッファーの長さ (バイト単位)。

flags
: 予約済み。ゼロに設定。

戻り値

成功した場合、メッセージまたはデータグラムの長さ (バイト単位) が戻されます。値 -1 はエラーを表します。値 `0` は、接続がクローズされていることを示します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
: 説明

EBADF
: *socket* が無効ソケット記述子です。

ECONNRESET
: 接続はピアによって強制的にクローズされました。

EFAULT
: *buf* および *len* パラメーターを使用すると、呼び出し元のアドレス・スペースの外側のストレージにアクセスすることになります。

EINVAL
: 要求が無効か、またはサポートされていません。MSG_OOB フラグが設定され、アウト・オブ・バンドのデータが使用できません。

ENOBUFS
: 使用可能なシステム・リソースが不十分で、呼び出しを完了させることができません。

ENOTCONN
: 接続されていないコネクション指向ソケットで受信しようとしました。

EOPNOTSUPP
: 指定したフラグは、このソケット・タイプまたはプロトコルの場合はサポートされていません。

ETIMEDOUT
接続の確立中またはアクティブな接続での伝送のタイムアウトのために、接続がタイムアウトになりました。

EWOULDBLOCK
socket は非ブロッキング・モードであり、データは読み取りに使用可能でありません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

スタックの特性

TCP/IP for z/VSE では、MSG_PEEK および MSG_OOB オプションはサポートされていません。

recvfrom() — ソケットでのメッセージ受信

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

int recvfrom(int                 socket,
             void               *buffer,
             size_t              length,
             int                 flags,
             struct sockaddr    *name,
             size_t             *namelen);
```

概要

recvfrom() 呼び出しは、記述子 *socket* によって指定されたソケットでデータを受信し、そのデータをバッファーに保管します。recvfrom() 呼び出しは、接続されているかどうかにかかわらず、どのソケットにも適用されます。

パラメーター
説明

socket ソケット記述子。

buffer データを受け取るバッファーへのポインター。

length *buffer* パラメーターが指すバッファーの長さ (バイト単位)。

flags 予約済み。ゼロに設定。

name データを受信するソケット・アドレス構造体へのポインター。*name* がゼロ以外の値の場合、送信元アドレスが戻されます。

namelen
name のサイズ (バイト単位)。

name がゼロ以外の場合は、メッセージの送信元アドレスが挿入されます。*namelen* は、最初に *name* と関連したバッファーのサイズに初期設定する必要がありますが、その後、戻りのときに変更され、バッファーに格納されているアドレスの実際のサイズを示します。

この呼び出しは、着信メッセージまたはデータの長さを戻します。データグラム・パケットが長すぎて、提供されたバッファーに収まらない場合には、データグラム・ソケットは、余分なバイトを廃棄します。ソケット *socket* でデータが使用できず、*socket* がブロッキング・モードの場合、`recvfrom()` 呼び出しはデータが到着するまで呼び出し元をブロックします。データが使用可能でなく、*socket* が非ブロッキング・モードの場合、`recvfrom()` は -1 を戻し、エラー・コードを EWOULDBLOCK に設定します。非ブロッキング・モードの設定方法の説明については、122 ページの『fcntl() — オープンされたソケット記述子の制御』または 180 ページの『ioctl() — ソケットの制御』を参照してください。

データグラム・ソケットの場合、この呼び出しは、データグラムが指定のバッファーに収まる場合は、送信されたデータグラム全体を戻します。ストリーム・ソケットは、データを分離する境界のない情報ストリームのように動作します。例えば、アプリケーション A および B がストリーム・ソケットと接続され、アプリケーション A が 1000 バイトを送信した場合には、この関数のそれぞれの呼び出しは、1 バイト、または 10 バイト、あるいは 1000 バイト全体を戻すことができます。したがって、ストリーム・ソケットを使用するアプリケーションは、この呼び出しをループに入れて、すべてのデータを受信するまで、この関数を呼び出す必要があります。

IPv6 のソケット・アドレス構造体: AF_INET6 ソケットの場合、アドレスは sockaddr_in6 アドレス構造体に返されます。sockaddr_in6 構造体は、ヘッダー・ファイル **in.h** の中に定義されます。

戻り値

成功した場合、メッセージまたはデータグラムの長さ (バイト単位) が戻されます。値 `0` は、接続がクローズされていることを示します。値 `-1` はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
: 説明

EBADF
: *socket* が無効ソケット記述子です。

ECONNRESET
: ピアによって接続が強制的にクローズされました。

EFAULT
: *buffer* および *length* パラメーターを使用すると、呼び出し元のアドレス・スペースの外側のストレージにアクセスすることになります。

EINVAL
: 要求が無効か、またはサポートされていません。MSG_OOB フラグが設定され、アウト・オブ・バンドのデータが使用できません。

ENOBUFS
: 使用可能なシステム・リソースが不十分で、呼び出しを完了させることができません。

ENOTCONN
: 接続されていないコネクション指向ソケットで受信しようとしました。

EOPNOTSUPP
指定したフラグは、このソケット・タイプの場合はサポートされません。

ETIMEDOUT
接続の確立中またはアクティブな接続での伝送のタイムアウトのために、接続がタイムアウトになりました。

EWOULDBLOCK
socket は非ブロッキング・モードであり、データは読み取りに使用可能でありません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

recvmsg() — ソケット上のメッセージの受信およびメッセージ・ヘッダーの配列への保管

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

ssize_t recvmsg(int socket, struct msghdr *msg,int flags);
```

概要

recvmsg() 呼び出しは、記述子 *socket* を用いてソケット上のメッセージを受信し、それをメッセージ・ヘッダーの配列に保管します。

パラメーター
説明

socket ソケット記述子。

msg メッセージを受け取るメッセージ・ヘッダーの配列。

flags *flags* パラメーターは、フラグを 1 つ以上指定することによって設定されます。複数のフラグを指定する場合は、論理 OR 演算子 (|) を使用してフラグを分離する必要があります。

メッセージ・ヘッダーは、**msghdr** 構造体によって定義されます。この構造体の定義は **socket.h** 組み込みファイルにあります。構造体の定義には、以下のエレメントが含まれます。

エレメント
説明

msg_iov
メッセージが中に入っている *iovec* バッファーの配列。

msg_iovlen
msg_iov 配列内のエレメントの数。

msg_name
送信側のアドレスが保管されているバッファーへのポインター。

msg_namelen
アドレス・バッファーのサイズ。

msg_control
補助データ、以下を参照。

msg_controllen
補助データ・バッファー長。

msg_flags
受信メッセージのフラグ。

対のシーケンスで構成される補助データです。それぞれ、データ配列とその後ろに続く **cmsghdr** 構造体から構成されます。データ配列には補助データ・メッセージが入り、**cmsghdr** 構造体にはアプリケーションが正しくデータを解析できるようにする記述情報が入ります。

エレメント
説明

cmsg_len
ヘッダーを含むデータ・バイト・カウント。

cmsg_level
送信元のプロトコル。

cmsg_type
プロトコル固有のタイプ。

socket.h ヘッダー・ファイルは、次のマクロを定義して、メッセージ・ヘッダーと関連した補助データのデータ配列へのアクセス権を取得します。

CMSG_DATA*(cmsg)*
引数が **cmsghdr** 構造体へのポインターである場合には、このマクロは、**cmsghdr** 構造体と関連したデータ配列への符号なしの文字ポインターを戻します。

CMSG_NXTHDR*(mhdr,cmsg)*
最初の引数が **msghdr** 構造体へのポインターであり、2 番目の引数が補助データの **cmsghdr** 構造体へのポインターである (その **msghdr** 構造体の **msg_control** フィールドによって指し示されている) 場合、このマクロは次の **cmsghdr** 構造体へのポインターを返します。この構造体が補助データの最後の **cmsghdr** である場合は、NULL ポインターを返します。

CMSG_FIRSTHDR*(mhdr)*
引数が **msghdr** 構造体へのポインターである場合、このマクロは、この **msghdr** 構造体に関連付けられた補助データの最初の **cmsghdr** 構造体へのポインターを返します。**msghdr** 構造体に関連付けられた補助データがない場合は、NULL ポインターを返します。

`recvmsg()` 呼び出しは、接続状態であるかどうかにかかわらず、ソケットに適用されます。

この呼び出しは、受信データの長さを戻します。ソケット *socket* でデータが使用できず、*socket* がブロッキング・モードの場合、`recvmsg()` 呼び出しはデータが到着

するまで呼び出し元をブロックします。データが使用できず、さらに *socket* が非ブロック・モードになっている場合、recvmsg() は -1 を戻し、エラー・コードを EWOULDBLOCK に設定します。

正常に終了した場合、メッセージ・ヘッダーの **msg_flags** メンバーは、受信メッセージで検出された条件を示すすべてのフラグのビット単位の包含 OR になります。

IPv6 のソケット・アドレス構造体**:** AF_INET6 ソケットの場合、アドレスは sockaddr_in6 アドレス構造体に返されます。sockaddr_in6 構造体は、ヘッダー・ファイル **in.h** の中に定義されます。

戻り値

成功した場合、メッセージの長さ (バイト単位) が戻されます。値 -1 はエラーを表します。具体的なエラーを errno の値が示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

select() — ソケットに関するアクティビティーのモニター

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <types.h>
#include <time.h>

int select(int             nmsgsfds,
           fd_set         *readlist,
           fd_set         *writelist,
           fd_set         *exceptlist,
           struct timeval *timeout);
```

概要

select() 呼び出しは、タイムアウトが起こるまで、ソケットの集合でのアクティビティーをモニターし、いずれかのソケットで、保留中の読み取り、書き込み、または例外処理条件がないかチェックします。

パラメーター

説明

num チェックされるソケット記述子の数。

アプリケーションがソケット 3、4、5、6、および 7 を割り振り、割り振られたそれらすべてをチェックしたい場合、指定したうち最も大きな記述子に 1 を加えた 8 を *num* に設定する必要があります。アプリケーションがソケット 3 および 4 をチェックする場合、*num* には 5 を設定します。

readlist,writelist,exceptlist

fd_set タイプ、メッセージ待ち行列 ID の配列、または sellist 構造体へのポインター (それぞれに、読み取り、書き込み、および例外条件の検査を行う)。渡すパラメーターのタイプは、ソケット記述子とメッセージ待ち行列 ID のどちらをモニターするのか、あるいは、両方をモニターするのかによ

って異なります。ソケット記述子をモニターするには、*nmsgsfds* の高位ハーフワードを 0 に、下位ハーフワードを (最大記述子 + 1) に設定し、fd_set ポインターを使用します。

timeout
: `select()` 呼び出しの完了を待機する時間へのポインター。

timeout は、NULL ポインターでない場合は、選択が完了するのを待機する最大インターバルを指定します。*timeout* が NULL ポインターの場合には、ソケットまたはメッセージが作動可能になるまで、`select()` 呼び出しはブロックします。ソケットをポーリングし、即時に戻るには、*timeout* が、ゼロ値に設定された *timeval* 構造体への非 NULL ポインターであることが必要です。

一度に複数のソケットをテストできるように、テストするソケットは、*fd_set* 型のビット・セット中に入れられます。ビット・セットは、複数のビットからなるストリングであり、*x* がセットのエレメントの場合は *x* を表すビットが 1 に設定され、*x* がセットのエレメントでない場合は *x* を表すビットが `0`に設定されます。例えば、ソケット 33 がビット・セットのエレメントである場合はビット 33 が 1 に設定され、ソケット 33 がビット・セットのエレメントでない場合はビット 33 が `0` に設定されます。

ビット・セットには、プロセスが割り振ることができるすべてのソケットのビットが含まれているので、ビット・セットのサイズは定数になります。プログラムで多数のソケットを割り振る必要がある場合には、ビット・セットのサイズを増やすことが必要なことがあります。ビット・セットのサイズの増加は、プログラムのコンパイル時に行う必要があります。ビット・セットのサイズを増やすには、*time.h* を含める前に FD_SETSIZE を定義します。 FD_SETSIZE は、プログラムで想定している `select()` を使用するソケットの最大値です。これは、*time.h* では 2048 に定義されます。ただし、TCP/IP for z/VSE は 8000 ソケットまで定義できます。

注:

1. _OPEN_MSGQ_EXT を定義して `select()` の拡張バージョンが使用される場合、FD_SETSIZE はアプリケーション・プログラムによってのみ定義することができます。sellist 構造体が使用される場合には、プログラムで FD_SETSIZE を「定義しないで」ください。
2. アプリケーション・プログラムが多数のソケット記述子を必要とする場合、そのような場合に起こりうるランタイム・エラーを回避するため、次のようなコーディングを行うべきです。
   - `select()` 呼び出しまたは `selectex()` 呼び出しの前に、*num* が FD_SETSIZE より大きいかどうかを調べるための検査を追加します。
   - コンパイル時に作成された静的ビット・ストリングに依存するのではなく、アプリケーション・プログラムの最大記述子の値を保持できる十分な大きさのビット・ストリングを動的に割り振ります。ユーザー自身のビット・ストリングの割り振り時に、`malloc()` を使用して、各ビットを表すのに十分な大きさのエリアを定義してください (次の 4 バイト倍数に切り上げ)。例えば、最大記述子値が 31 の場合には、4 バイトが必要です。最大記述子値が 32 の場合には、8 バイトが必要です。
   - ユーザー自身のビット・ストリングを動的に割り振ると、FD_ZERO() マクロは動作しません。アプリケーションでは、memset 関数、すなわち、

memset(*ptr*,*0*,*mallocsize*) を使用して、そのストレージをゼロにする必要があります。その他のマクロは、操作中のビット記述子がビット・ストリング内にある限り、動的に割り振られたビット・ストリングで使用できます。記述子数がビット・ストリングより大きいと、予測不可能な結果が起こる場合があります。

アプリケーション・プログラムは、*readlist*、*writelist*、および *exceptlist* パラメーターが、*num* パラメーターに指定されたストリング・サイズと同じ大きさのビット・ストリングを示さなければなりません。TCP/IP サービスは、それぞれのビット・ストリングのビット 0 から *num-1*-1 にアクセスを試みます。ビット・ストリングが短すぎると、アプリケーション・プログラムの実行時に、予測できない結果を受け取ることになります。

ビット・セットを操作するために、以下のマクロが提供されています。

マクロ 説明

FD_ZERO(&*fdset*)
ビット・セット *fdset* のすべてのビットをゼロに設定します。この操作の後では、ソケットはビット・セットにエレメントとして含まれません。このマクロが呼び出されるのは、ソケットをメンバーとして設定する FD_SET() を呼び出す前に、ビット・セットを初期化する場合です。

注: malloc() を使用して動的に新しいエリアを割り振る場合、FD_ZERO() マクロにより予測できない結果となることがあるので、使用しないでください。memset() 関数を使用して、エリアをゼロにする必要があります。

FD_SET(*sock*, &*fdset*)
ソケット *sock* のビットを 1 に設定して、*sock* をビット・セット *fdset* のメンバーにします。

FD_CLR(*sock*, &*fdset*)
ビット・セット *fdset* のソケット *sock* のビットをクリアします。この操作により、該当するビットがゼロに設定されます。

FD_ISSET(*sock*, &*fdset*)
sock が、ビット・セット *fdset* のメンバーの場合には、ゼロ以外を戻します。*sock* が *fdset* のメンバーではない場合には、ゼロを戻します (この操作は、*sock* を表すビットを含む 32 ビット値を戻します)。

nmsgsfds パラメーターと select() からの戻り値を操作するために、以下のマクロが提供されています。

マクロ 説明

_SET_FDS_MSGS(*nmsgsfds, nmsgs, nfds*)
nmsgsfds の高位ハーフワードを *nmsgs* に設定し、*nmsgsfds* の下位ハーフワードを *nfds* に設定します。

_NFDS(*n*)
select() からの戻り値 *n* が負ではない場合には、読み取り、書き込み、および例外の基準を満たす記述子の数を戻します。1 つの記述子でも、それが複数の指定基準を満たす場合には、複数回カウントされる場合があります。

_NMSGS(*n*)
: `select()` からの戻り値 *n* が負ではない場合には、読み取り、書き込み、および例外の基準を満たすメッセージ待ち行列の数を戻します。

ソケットは、着信データがそのためにバッファーに入れられるか、あるいは接続要求が保留中のときに、読み取りの作動が可能になります。ソケットのいずれかが読み取り可能になっているかどうかをテストするには、関数が動的に割り振られていた場合は、FD_ZERO() または `memset()` を使って *readlist* の fdset ビット・セットを初期化して、テストする各ソケットごとに FD_SET() を呼び出します。

発信データ用のバッファー・スペースがある場合には、ソケットは書き込み可能です。`connect()` が完了すると、接続プロセス中の (`connect()` が EINPROGRESS を戻した) 非ブロッキングのストリーム・ソケットは、選択され、書き込みできる状態になります。データ量がバッファー・スペース量より少ないと、`write()`、`send()`、または `sendto()` への呼び出しはブロックしません。ソケットのいずれかが書き込み可能な状態かどうかをテストするには、関数が動的に割り振られていた場合は FD_ZERO() または `memset()` を使って *writelist* 内の fdset ビット・セットを初期化して、テストする各ソケットに対して FD_SET() を使用します。

プログラマーは、*readlist*、*writelist*、および *exceptlist* パラメーターのいずれかの NULL を渡すことができます。ただし、これらが NULL ではないときには、すべて同じ型の構造体を示す必要があります。

`select()` に渡されるソケットの集合はビット・セットなので、ソケットの状態を調べるためにソケットをポーリングする前に、`select()` 呼び出しで各ビット・セット内のそれぞれのビットをテストする必要があります。`select()` 呼び出しでテストされるのは、範囲が `0` から *num*-1 のソケットだけです。

戻り値

値 -1 は、エラー・コードをチェックしてエラーがあるか確認する必要があることを示します。値ゼロは、時間制限が満了であることを示します。

戻り値が `0` より大きい場合には、高位 16 ビットでメッセージ待ち行列が指定され、下位 16 ビットで記述子数が指定されるので、この戻り値は *nmsgsfds* と類似しています。高位 16 ビットがメッセージ待ち行列に関連した数を指定し、下位 16 ビットがファイル記述子に関連した数を指定します。これらの値は、読み取り、書き込み、および例外のそれぞれの基準を満たす合計値を示します。ソケット記述子の戻り値が 65,535 を超える場合、65,535 のみが報告されます。

戻り値がゼロより大きい場合には、各ビット・セットで作動可能なソケットが 1 に設定されます。作動不能の各ビット・セットのソケットは、ゼロに設定されます。各ソケットの状況をテストするには、そのソケットでマクロ FD_ISSET() を使用します。

エラー・コード
: 説明

EBADF
: ビット・セットの 1 つが無効ソケットを指定したか、あるいはメッセージ待ち行列 ID が無効です。ソケットが設定される前に、ビット・セットをクリアする FD_ZERO() は呼び出されなかったと考えられます。

EFAULT
パラメーターの 1 つに、無効アドレスが含まれていました。

EINVAL
timeval 構造体のフィールドの 1 つが無効か、あるいは無効 *nmsgsfds* 値がありました。

EIO ネットワークの問題が原因で、選択されるソケットの 1 つが操作不能です。これは、TCP/IP がシャットダウンされている場合に、ソケットについて発生する可能性があります。各記述子ごとに個別の select() が待機せずに障害が発生するまでループするようにコーディングすることによって、どの記述子がよくないのかを判別できます。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EIO に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

例

以下に示すのは、select() 呼び出しの例です。

```
#define _OPEN_MSGQ_EX        /* needed for _SET_FDS_MSGS macro */
#include <time.h>
#include <types.h>
#include <stdio.h>

/* This function returns
  -1 if an error occurred
   0 if aSocket is NOT ready for read
   1 if aSocket is ready for read.
 */
int testSocketReadyForRead(int aSocket)
{
  fd_set socketSet;
  struct timeval timeout;
  int rc, number;

  /* Initialize timeout structure. */
  timeout.tv_sec=1;     */ seconds */

  /* Initialize socket set bits and add sockets to be examined. */
  FD_ZERO(&socketSet)
  FD_SET(aSocket, &socketSet);

  /* Set the number parameter. */
  _SET_FDS_MSGS(number,
                0, /* don't monitor message queues */
                aSocket+1);

  /* check for READ availability on this socket */
  rc=select(number,
            &socketSet,  /* set of sockets to check for readability */
            NULL, /* set of sockets to check whether ready to write */
            NULL, /* set of sockets to check for pending exceptions */
            &timeout);
  if (rc<0)
  { perror("select");
    return rc;
  }
  else return (FD_ISSET(aSocket,&socketSet) != 0);
}
```

selectex() — ソケットに関するアクティビティーのモニター

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#define _ALL_SOURCE
#include <types.h>
#include <time.h>

int selectex( int             nmsgsfds,
              fd_set         *readlist,
              fd_set         *writelist,
              fd_set         *exceptlist,
              struct timeval *timeout,
              int            *ecbptr);
```

概要

selectex() 呼び出しは、*readlist*、*writelist*、または *exceptlist* では記述されないイベントを定義する ECB を使用できるように、select() 呼び出しを拡張したものです。

selectex() 呼び出しは、タイムアウトになるか、または ECB がポストされるまで、ソケットの集合でのアクティビティーをモニターして、いずれかのソケットで読み取り、書き込み、または保留中の例外処理条件があるかどうかをチェックします。

詳細は、select() を参照してください。

パラメーター
: 説明

num
: チェックされるソケット記述子の数。(このパラメーター、および以下のようなその他のパラメーターの十分な説明については、select() を参照してください)。

readlist
: 読み取りをチェックするための fd_set タイプへのポインター。

writelist
: 書き込みをチェックするための fd_set タイプへのポインター。

exceptlist
: 保留中の例外条件をチェックするための fd_set タイプへのポインター。

timeout
: selectex() 呼び出しの完了を待機する時間へのポインター。

ecbptr
: この変数に、次の値の 1 つを入れることができます。

1. ユーザー・イベント制御ブロックへのポインター。この *ecbptr* の使用を指定するには、高位ビットを '0'B に設定する必要があります。
2. 複数の ECB ポインターからなるリストへのポインター。この *ecbptr* の使用を指定するには、高位ビットを '1'B に設定する必要があります。

   リストには、最大 254 ECB までポインターを入れることができます。リストの最後のポインターの高位ビットは、'1'B に設定する必要があります。
3. NULL ポインター。ECB が指定されていないことを示します。

戻り値

値 -1 は、エラー・コードをチェックしてエラーがあるか確認する必要があることを示します。値 0 は、時間制限が満了になったこと、または ECB がポストされたことを示します。

0 より大きい戻り値は、読み取り、書き込み、および例外の基準に一致した合計を示します。1 つの記述子が要求された複数の基準を満たす場合には、記述子が複数回カウントされる場合があります。ソケット記述子の戻り値が 65,535 を超える場合、65,535 のみが報告されます。

戻り値がゼロより大きい場合には、各ビット・セットで作動可能なソケットが 1 に設定されます。作動不能の各ビット・セットのソケットは、ゼロに設定されます。各ソケットの状況をテストするには、そのソケットでマクロ FD_ISSET() を使用します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EIO に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

send() — ソケットでのデータ送信

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

ssize_t send(int socket, const void *msg, size_t length, int flags);
```

概要

send() 呼び出しは、記述子が *socket* のソケットでデータを送信します。send() 呼び出しは、すべての接続されたソケットに適用されます。

パラメーター
　説明

socket
　ソケット記述子。

msg 送信するメッセージが入っているバッファーへのポインター。

length *msg* パラメーターで示される、メッセージの長さ。APAR PQ55591 の PTF がインストールされていない場合、指定される最大バイト数は 64 K です。

flags 予約済み。ゼロに設定。

送信されるソケット・データを収容するのに使用可能なバッファー・スペースが十分でなく、ソケットがブロッキング・モードの場合、send() は追加バッファー・スペースが使用可能になるまで呼び出し元をブロックします。ソケットが非ブロッキング・モードの場合、send() は -1 を戻し、エラー・コードを EWOULDBLOCK に設定します。非ブロッキング・モードの設定方法の説明については、122 ページの『fcntl() — オープンされたソケット記述子の制御』または 180 ページの『ioctl() — ソケットの制御』を参照してください。

データグラム・ソケットの場合、データグラムが TCP/IP バッファーに収まれば、この呼び出しはデータグラム全体を送信します。ストリーム・ソケットは、データを分離する境界のない情報ストリームのように動作します。例えば、アプリケーションで 1000 バイトを送信したい場合には、この関数へのそれぞれの呼び出しによって、1 バイト、または 10 バイト、あるいは 1000 バイト全体が送信できます。したがって、ストリーム・ソケットを使用するアプリケーションは、この呼び出しをループに入れて、すべてのデータが送信されてしまうまで、この関数を呼び出す必要があります。

戻り値

値 -1 は。ローカルに検出されたエラーを表します。エラー・コードの値によって、具体的なエラーを示します。send() ルーチンでは、送信の失敗が暗黙的に示されることはありません。

0 またはそれより大きい値は、送信されたバイト数を示しています。しかし、これはデータの送達が完了したことを保証するわけではありません。

エラー・コード
: 説明

EBADF
: *socket* が無効ソケット記述子です。

ECONNRESET
: 接続はピアによって強制的にクローズされました。

EDESTADDRREQ
: ソケットはコネクション指向でなく、ピア・アドレスが設定されていません。

EFAULT
: *msg* および *length* パラメーターを使用すると、呼び出し元のアドレス・スペースの外側のストレージにアクセスすることになります。

ENOBUFS
: メッセージの送信にバッファー・スペースを使用することができません。

ENOTCONN
: ソケットが接続していません。

EOPNOTSUPP
: *socket* 引数と関連付けられているソケットは、*flags* に設定されている 1 つ以上の値をサポートしていません。

EWOULDBLOCK
: *socket* は非ブロッキング・モードであり、使用可能なデータ・バッファーがありません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

スタックの特性

TCP/IP for z/VSE では、MSG_OOB および MSG_DONTROUTE オプションはサポートされていません。

sendmsg() — ソケットでのメッセージ送信

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

ssize_t sendmsg(int socket, struct msghdr *msg, int flags);
```

概要

sendmsg() 呼び出しは、メッセージ・ヘッダーの配列で渡されたメッセージを記述子 *socket* のソケット上で送信します。

パラメーター
: 説明

socket
: ソケット記述子。

msg
: メッセージがそこから送信されるメッセージ・ヘッダーの配列。

flags
: *flags* パラメーターを設定するには、0 を指定するか、1 つ以上のフラグを指定します。複数のフラグを指定する場合は、論理 OR 演算子 (|) を使用してフラグを分離する必要があります。

  メッセージ・ヘッダーは **msghdr** 構造体によって定義されます。この構造体は、**socket.h** 組み込みファイルにあり、以下のエレメントを含みます。

  エレメント
  : 説明

  msg_iov
  : メッセージが中に入っている *iovec* バッファーの配列。

  msg_iovlen
  : *msg_iov* 配列内のエレメントの数。

  msg_name
  : 受信側のアドレスが入っているバッファーへのポインター。

  msg_namelen
  : アドレス・バッファーのサイズ。

  msg_control
  : 補助データ、以下を参照。

  msg_controllen
  : 補助データ・バッファー長。

  msg_flags
  : 受信メッセージのフラグ。

  対のシーケンスで構成される補助データです。それぞれ、データ配列とその後ろに続く **cmsghdr** 構造体から構成されます。データ配列には補助デー

タ・メッセージが入り、**cmsghdr** 構造体にはアプリケーションが正しくデータを解析できるようにする記述情報が入ります。

socket.h ヘッダー・ファイルは、少なくとも以下のメンバーを含んでいる **cmsghdr** 構造体を定義します。

エレメント
: 説明

cmsg_len
: ヘッダーを含むデータ・バイト・カウント。

cmsg_level
: 送信元のプロトコル。

cmsg_type
: プロトコル固有のタイプ。

socket.h ヘッダー・ファイルは、次のマクロを定義して、メッセージ・ヘッダーと関連した補助データのデータ配列へのアクセス権を取得します。

CMSG_DATA*(cmsg)*
: 引数が **cmsghdr** 構造体へのポインターである場合には、このマクロは、**cmsghdr** 構造体と関連したデータ配列への符号なしの文字ポインターを戻します。

CMSG_NXTHDR*(mhdr,cmsg)*
: 最初の引数が **msghdr** 構造体へのポインターであり、2 番目の引数が補助データの **cmsghdr** 構造体へのポインターである (その **msghdr** 構造体の **msg_control** フィールドによって指し示されている) 場合、このマクロは次の **cmsghdr** 構造体へのポインターを返します。この構造体が補助データの最後の **cmsghdr** である場合は、NULL ポインターを返します。

CMSG_FIRSTHDR*(mhdr)*
: 引数が **msghdr** 構造体へのポインターである場合、このマクロは、この **msghdr** 構造体に関連付けられた補助データの最初の **cmsghdr** 構造体へのポインターを返します。**msghdr** 構造体に関連付けられた補助データがない場合は、NULL ポインターを返します。

`sendmsg()` 呼び出しは、接続状態であるかどうかにかかわらず、ソケットに適用されます。

この呼び出しは、送信データの長さを戻します。送信されるソケット・データを入れるのに十分なバッファー・スペースが使用可能でなく、ソケットがブロッキング・モードの場合、`sendmsg()` は追加バッファー・スペースが使用可能になるまで呼び出し元をブロックします。ソケットが非ブロッキング・モードの場合、`sendmsg()` は -1 を戻し、エラー・コードを EWOULDBLOCK に設定します。

IPv6 のソケット・アドレス構造体**:** AF_INET6 ソケットでは、msg_name が指定された場合、アドレスは sockaddr_in6 アドレス構造体にある必要があります。sockaddr_in6 構造体は、ヘッダー・ファイル **in.h** の中に定義されます。

戻り値

成功した場合、メッセージの長さ (バイト単位) が戻されます。値 -1 はエラーを表します。具体的なエラーを errno の値が示します。

0 またはそれより大きい値は、送信されたバイト数を示していますが、これはデータの送達が完了したことを保証するわけではありません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

sendto() — ソケットでのデータ送信

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

ssize_t sendto(int                    socket,
               const void            *msg,
               size_t                 length,
               int                    flags,
               const struct sockaddr *address,
               size_t                 address_length);
```

概要

sendto() 呼び出しは、記述子が *socket* のソケットでデータを送信します。
sendto() 呼び出しは、接続または非接続ソケットのどちらかに適用されます。

パラメーター
　説明

socket
　ソケット記述子。

msg 送信するメッセージが入っているバッファーへのポインター。

length *msg* パラメーターによって指されたバッファー内のメッセージの長さ。APAR PQ55591 の PTF がインストールされていない場合、指定される最大バイト数は 64 K です。

flags 予約済み。ゼロに設定。

address
　ターゲットのアドレス。

address_length
　address が指すアドレスのサイズ。

送信されるソケット・データを入れるのに十分なバッファー・スペースが使用可能でなく、ソケットがブロッキング・モードの場合、sendto() は追加バッファー・スペースが使用可能になるまで呼び出し元をブロックします。ソケットが非ブロッキング・モードの場合、sendto() は -1 を戻し、エラー・コードを EWOULDBLOCK に設定します。非ブロッキング・モードの設定方法の説明につい

ては、 122 ページの『fcntl() — オープンされたソケット記述子の制御』または 180 ページの『ioctl() — ソケットの制御』を参照してください。

データグラム・ソケットの場合、データグラムが TCP/IP バッファーに収まれば、この呼び出しはデータグラム全体を送信します。ストリーム・ソケットは、データを分離する境界のない情報ストリームのように動作します。例えば、アプリケーションで 1000 バイトを送信したい場合には、この関数へのそれぞれの呼び出しによって、1 バイト、または 10 バイト、あるいは 1000 バイト全体が送信できます。したがって、ストリーム・ソケットを使用するアプリケーションは、この呼び出しをループに入れて、すべてのデータが送信されてしまうまで、この関数を呼び出す必要があります。

IPv6 用ソケット・アドレス構造体**:** sockaddr_in6 構造体は `in.h` ヘッダーに追加されます。これは、IPv6 特定アドレスの、アプリケーションとシステムの間のパスに使用されます。

戻り値

成功した場合、送信された文字数が戻されます。値 -1 はエラーを表します。具体的なエラーを *errno* の値が示します。この呼び出しをデータグラム・ソケットに対して使用した場合、送信の失敗が戻り値で暗黙的に示されることはありません。

`0` またはそれより大きい値は、送信されたバイト数を示していますが、これはデータの送達が完了したことを保証するわけではありません。

エラー・コード
: 説明

EAFNOSUPPORT
: アドレス・ファミリーがサポートされていません (これは AF_INET または AF_INET6 ではありません)。

EBADF
: *socket* が無効ソケット記述子です。

ECONNRESET
: 接続はピアによって強制的にクローズされました。

EFAULT
: *msg* および *length* パラメーターを使用すると、呼び出し元のアドレス・スペースの外側のストレージにアクセスすることになります。

EINVAL
: *address_length* が、指定アドレス・ファミリーの有効アドレスのサイズではありません。

ENOBUFS
: メッセージの送信にバッファー・スペースを使用することができません。

ENOTCONN
: ソケットが接続していません。

EOPNOTSUPP
: *socket* 引数と関連付けられているソケットは、*flags* に設定されている 1 つ以上の値をサポートしていません。

EWOULDBLOCK
socket は非ブロッキング・モードであり、使用可能なデータ・バッファーがありません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、*errno* は EOPNOTSUPP に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

sethostent() — ホスト情報データ・セットのオープン

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

void sethostent(int stayopen);
```

概要

sethostent() 呼び出しは、既知のホストに関する情報を含むデータ・セットをオープンして巻き戻します。*stayopen* フラグがゼロ以外の場合、データ・セットは各呼び出しの後、オープンされたままになります。

戻り値

sethostent() は、値を戻しません。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

setibmopt() - IBM TCP/IP イメージの設定

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <socket.h>


int setibmopt(int cmd, struct ibm_tcpimage *bfrp);
```

概要

setibmopt() 関数呼び出しを使用して、TCP/IP オプションを設定します。現在サポートされているコマンドは、IBMTCP_IMAGE だけです。このコマンドによって、setibmopt() は、アプリケーションが接続されるアクティブな TCP/IP イメージ・スタックを選択できます。

ibm_tcpimage をなにも選択されていない状態にリセットするためには、*name* をすべてブランクに設定してください。

パラメーター
説明

cmd *cmd* の値は、実行するコマンドに設定する必要があります。現在、IBMTCP_IMAGE だけがサポートされており、説明したとおり、*bfrp* パラメーターと対にする必要があります。

bfrp ibm_tcpimage 構造体へのポインター。

TCP/IP イメージをソケットに設定するには、アプリケーションは、値を以下のように ibm_tpcimage 構造体に設定する必要があります。

エレメント
: 説明

状況 0 は、未確認、検査不要を意味します。現在では、これは意味を持つ唯一の値です。

バージョン
: 0 は、分かっている場合は、戻る時点で設定されるという意味です。

名前 名前は、左そろえされている大文字であり、ブランクが埋め込まれている必要があります。また、アクティブ TCP スタックの名前でなければなりません。

戻り値

正常に実行された場合、setibmopt() は 0 を返します。正常に実行されなかった場合、setibmopt() は -1 を返し、errno を以下のいずれかの値に設定します。

エラー・コード
: 説明

EFAULT
: 提供済み *bfrp* を使用した結果、アクセス不能な保管場所へアクセスしています。

EIBMBADTCPNAME
: PFS の名前は、構成済みではないか、またはソケット PFS ではないことが指定されました。

EOPNOTSUPP
: *cmd* 関数はサポートされてません。

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

setnetent() — ネットワーク情報データ・セットのオープン

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

void setnetent(int stayopen);
```

概要

setnetent() 呼び出しは、既知のネットワークに関する情報を含むデータ・セットをオープンして巻き戻します。*stayopen* フラグがゼロ以外の場合、各 setnetent()

呼び出しの後、データ・セットはオープンされたままになります。

戻り値

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

setprotoent() — プロトコル情報データ・セットのオープン

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

void setprotoent(int stayopen);
```

概要

setprotoent() 呼び出しは、既知のプロトコルに関する情報を含むデータ・セットをオープンして巻き戻します。*stayopen* フラグがゼロ以外の場合、データ・セットは各呼び出しの後、オープンされたままになります。

戻り値

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

setservent() — ネットワーク・サービス情報データ・セットのオープン

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <netdb.h>

void setservent(int stayopen);
```

概要

setservent() 呼び出しは、既知のサービスに関する情報を含むデータ・セットをオープンして巻き戻します。*stayopen* フラグがゼロ以外の場合、各 setservent() 呼び出しの後、データ・セットはオープンされたままになります。

戻り値

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

setsockopt() — ソケットに関連したオプションの設定

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

int setsockopt(int         socket,
               int         level,
               int         option_name,
               const void *option value,
               size_t      option_length);
```

IPv6: IPv6 ソケット・オプションに対するサポートを組み込むには、次のコードを追加してください。

```
#define _OPEN_SYS_SOCK_IPV6 1
#include <in.h>
```

概要

setsockopt() 呼び出しは、ソケットと関連したオプションを設定します。オプションは、複数のプロトコル・レベルで指定できます。最上位のソケット・レベルには必ず存在します。

パラメーター
: 説明

socket
: ソケット記述子。

level
: オプションが設定中のレベル。

option_name
: 指定ソケット・オプションの名前。

option_value
: オプション・データへのポインター。

option_length
: オプション・データの長さ。

ソケット・オプションを操作する際、オプションがどのレベルにあるのかとオプション名を指定する必要があります。ソケット・レベルのオプションを操作するため、 *level* パラメーターを **socket.h** に定義されているように SOL_SOCKET に設定する必要があります。 IPv4 または IPv6 レベルでオプションを操作するには、level パラメーターを、**socket.h** の定義に従って IPPROTO_IP に設定するか、または **in.h** の定義に従って IPPROTO_IPV6 に設定する必要があります。

option_value および *option_length* パラメーターが、特定の設定コマンドで使用されるデータを渡すために使用されます。*option_value* パラメーターは、設定コマンドで必要なデータを含むバッファーを指しています。*option_value* パラメーターはオプションで、コマンドでデータが不要な場合には、NULL ポインターに設定できます。*option_length* パラメーターは、*option_value* によって示されるデータのサイズに設定する必要があります。

SO_LINGER、SO_RCVTIMEO、および SO_SNDTIMEO を除くソケット・レベルのすべてのオプションでは、*option_value* が整数を指していて、*option_length* には整数のサイズが設定されていると予期されます。整数がゼロ以外の場合には、オプ

ションは使用可能です。ゼロの場合は、オプションは使用不可です。SO_LINGER オプションの場合は、*option_value* は、**socket.h** で定義されている **linger** 構造体を示していると想定されます。この構造体の定義を以下の例に示します。

```
struct  linger
{
        int     l_onoff;                /* option on/off */
        int     l_linger;               /* linger time */
};
```

SO_LINGER オプションが使用不可の場合、*l_onoff* フィールドは 0 に設定されます。ゼロ以外の値の場合、オプションは使用可能になります。*l_linger* フィールドは、クローズ時に残っている時間の長さを指定します。 *l_linger* の単位は秒です。

以下のようなオプションが、ソケット・レベルで認識されます。

オプション
: 説明

SO_LINGER
: データが存在している場合、クローズ時にすぐに戻りません。このオプションが使用可能に設定されている場合、close() が呼び出されたときに未送信データがあると、データが送信されるか、または接続がタイムアウトになるまで、close() 呼び出しの間、呼び出し側アプリケーション・プログラムはブロックされます。このオプションが使用不可にされている場合、TCP/IP アドレス・スペースは、データを送信しようとするのを待ちます。データ転送は通常は成功しますが、TCP/IP アドレス・スペースがデータを送信しようとするのを待つ時間は限られているので、成功が保証されるわけではありません。close() 呼び出しは、呼び出し元をブロックしないで、戻ります。このオプションは、ストリーム・ソケットにのみ意味を持ちます。

SO_KEEPALIVE
: このオプションは、ソースの互換性のためにのみ用意されています。何もアクションは行いませんが、この代わりに、ユーザーは一般的 TCP/IP 設定である SET PULSE_TIME=nnn を使用するべきです。この TCP/IP オプションは TCP/IP パーティション全部に効果を及ぼしますが、それは SO_KEEPALIVE が単一の TCP 接続に与える効果と同じです。

SO_REUSEADDR
: このオプションは、ソースの互換性のためにのみ用意されています。何もアクションは行いませんが、TCP/IP は、暗黙にアドレスの即時再使用を許可しています。

以下のオプションが、IPv4 レベルで認識されます。

オプション
: 説明

IP_ADD_MEMBERSHIP
: このオプションを使用すると、特定のインターフェース上にあるマルチキャスト・グループを結合することができます (インターフェースをこのオプションで指定する必要があります)。マルチキャスト・データグラムを受信するアプリケーションのみが、マルチキャスト・グループを結合する必要があります。送信のみを行うアプリケーションは、マルチキャスト・グループを結合する必要はありません。

マルチキャスト IP アドレスとインターフェース IP アドレスは、**in.h** 内の使用可能な以下の構造体に渡されます。

```
struct ip_mreq
{
   struct in_addr imr_multiaddr; /* IP multicast addr of group */
   struct in_addr imr_interface; /* local IP addr of interface */
};
```

渡された mreq 構造体のインターフェース・アドレスに INADDR_ANY が指定されていると、次のようにデフォルトのインターフェースが選択されます。

- mreq 構造体内に指定されているグループ・アドレスが、GATEWAY ステートメントに指定されている場合は、そのインターフェースを使用します。
- GATEWAY ステートメントに 224.0.0.0 が指定されている場合は、そのインターフェースを使用します。
- DEFAULTNET が指定されており、それにマルチキャスト機能がある場合は、そのインターフェースを使用します。

IP_ADD_SOURCE_MEMBERSHIP

このオプションを使用すると、ip_mreq_source 構造体によって指定された source-specific マルチキャスト・グループを結合することができます。ip_mreq_source 構造体は、**in.h** で定義されます。

IP_BLOCK_SOURCE

このオプションを使用すると、与えられたソースから与えられたマルチキャスト・グループへの動きをブロックできます (例えば、ユーザーがそのソースを「mute」した場合)。ソース・マルチキャスト・グループは、**in.h** で定義されている ip_mreq_source 構造体によって指定されます。

IP_DROP_MEMBERSHIP

このオプションを使用すると、マルチキャスト・グループをそのままにしておくことができます。

マルチキャスト IP アドレスとインターフェース IP アドレスは、**in.h** 内の使用可能な以下の構造体に渡されます。

```
struct ip_mreq
{
   struct in_addr imr_multiaddr; /* IP multicast addr of group */
   struct in_addr imr_interface; /* local IP addr of interface */
};
```

渡された mreq 構造体のインターフェース・アドレスに INADDR_ANY が指定されていると、システムはインターフェースに関係なく、グループ (クラス D) アドレスが一致する最初のグループを削除します。

IP_DROP_SOURCE_MEMBERSHIP

このオプションを使用すると、ip_mreq_source 構造体によって指定された source-specific マルチキャスト・グループをそのままにしておくことができます。ip_mreq_source 構造体は、**in.h** で定義されます。

IP_MULTICAST_IF

このソケット・アプリケーションからアウトバウンド・マルチキャスト・データグラムを送信するためのインターフェースを設定します。マルチキャス

ト・データグラムは、一度に 1 つのインターフェースでのみ送信されます。IP アドレスは struct in_addr を使用して渡されます。

渡されたインターフェース・アドレスに INADDR_ANY が指定されている場合は、次のようにデフォルトのインターフェースが選択されます。

- GATEWAY ステートメントに 224.0.0.0 が指定されている場合は、そのインターフェースを使用します。
- DEFAULTNET が指定されており、それにマルチキャスト機能がある場合は、そのインターフェースを使用します。

IP_MULTICAST_LOOP
発信マルチキャスト・データグラムのループバックを使用可能/使用不可にします。デフォルトでは、使用可能です。使用可能な場合、発信マルチキャスト・グループを結合しているマルチキャスト・アプリケーションは、そのアドレス / ポートの対に宛先指定されているマルチキャスト・データグラムのコピーを受け取ることができます。ループバック標識は `u_char` として渡されます。ループバックが使用不可の場合は、0 が指定されます。ループバックが使用可能の場合は、1 が指定されます。

IP_MULTICAST_TTL
発信マルチキャスト・データグラムの IP データグラム存続時間 (ホップ) を設定します。デフォルト値は 1 です (つまり、ローカル・サブネットへのマルチキャストだけです)。TTL 値は `u_char` として渡されます。

IP_UNBLOCK_SOURCE
このオプションは、IP_BLOCK_SOURCE オプションを使用して実行された操作を取り消すために使用します (例えば、ユーザーがそのソースを「mutes」した場合)。ソース・グループは、**in.h** で定義されている ip_mreq_source 構造体によって指定されます。

MCAST_BLOCK_SOURCE
このオプションを使用すると、与えられたソースから与えられたグループへのデータをブロックできます (例えば、ユーザーがそのソースを「mutes」した場合)。ソースは、**in.h** で定義されている group_source_req 構造体によって指定されます。

MCAST_JOIN_GROUP
このオプションを使用すると、any-source グループを結合することができます。グループは group_req 構造体よって指定されます。group_req 構造体は **in.h** で定義されます。

MCAST_JOIN_SOURCE_GROUP
このオプションを使用すると、source-specific グループを結合することができます。ソースは、**in.h** で定義されている group_source_req 構造体によって指定されます。

MCAST_LEAVE_GROUP
このオプションを使用すると、any-source グループをそのままにしておくことができます。グループは group_req 構造体よって指定されます。group_req 構造体は **in.h** で定義されます。

MCAST_LEAVE_SOURCE_GROUP
このオプションを使用すると、source-specific グループをそのままにしておくことができます。ソースは、**in.h** で定義されている group_source_req 構造体によって指定されます。

MCAST_UNBLOCK_SOURCE
このオプションを使用すると、MCAST_BLOCK_SOURCE オプションを使用して実行された操作を取り消します (例えば、ユーザーがそのソースを後に「unmutes」した場合)。ソースは、**in.h** で定義されている group_source_req 構造体によって指定されます。

以下のオプションが、IPv6 レベルで認識されます。

オプション
説明

IPV6_JOIN_GROUP
渡された ipv6_mreq 構造体に指定されたマルチキャスト・グループに参加することで、マルチキャスト・パケットの受信を制御します。ipv6_mreq 構造体は **in.h** で定義されます。

IPV6_LEAVE_GROUP
パス済みの ipv6_mreq 構造体に指定されるマルチキャスト・グループから離れることにより、マルチキャスト・パケットの受信を制御します。ipv6_mreq 構造体は **in.h** で定義されます。

IPV6_MULTICAST_HOPS
発信マルチキャスト・パケットのホップ限界を設定します。ホップ限界値は `int` として渡されます。

IPV6_MULTICAST_IF
発信マルチキャスト・パケットのインターフェースを設定します。インターフェースの指定には、インターフェース索引が使用されます。これは `u_int` として渡されます。

IPV6_MULTICAST_LOOP
送信ホスト自体が (発信インターフェース上で) 所属するグループに対してマルチキャスト・データグラムが送信された場合、このオプションが 1 に設定されていると、ローカル配信のために、IP レイヤーによってデータグラムのコピーがループバックされます。このオプションがゼロに設定されていた場合は、コピーはループバックされません。その他のオプション値は、EINVAL の errno を戻します。デフォルトでは 1 (ループバック) です。オプション値は `int` として渡されます。

IPV6_UNICAST_HOPS
発信ユニキャスト・パケットのホップ限界を制御します。ホップ限界値は `int` として渡されます。

IPV6_V6ONLY
ソケットが IPv6 コミュニケーションのみに制限されているかどうかを判別します。デフォルト設定は off です。オプション値は `int` として渡されます。ゼロ以外の値は、オプションが使用可能 (ソケットは IPv6 コミュニケーションのみに使用可能) であることを示します。0 は、オプションが使用不可であることを示します。

注: これらのオプションを使用するには、フィーチャー・テスト・マクロ `#define _OPEN_SYS_SOCK_IPV6` を使用する必要があります。

戻り値

値 `0` は成功を表します。値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
: 説明

EBADF
: *socket* パラメーターが無効ソケット記述子です。

EFAULT
: *option_value* および *option_length* パラメーターを使用すると、呼び出し元のアドレス・スペースの外側のストレージにアクセスすることになります。

EINVAL
: 指定オプションが指定ソケット・レベルで無効であるか、またはソケットがシャットダウンされていました。

ENOBUFS
: 使用可能なシステム・リソースが不十分で、呼び出しを完了させることができません。

ENOPROTOOPT
: *option_name* パラメーターが認識されていないか、または *level* パラメーターが SOL_SOCKET ではありません。

ENOSYS
: 関数がインプリメントされていません。まだ使用可能になっていない関数を使用しようとしました。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、`errno` は ENOSYS に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

スタックの特性

TCP/IP for z/VSE ではオプション SO_LINGER のみがサポートされています。SO_KEEPALIVE および SO_REUSEADDR に対するエミュレーション・サポートも認可されます。

- SO_KEEPALIVE

  このオプションは、ソース・コード互換性のためにのみサポートされています。実際、キープアライブ値を設定しても、TCP 接続には何も影響はありません。その代わりに、ユーザーは SET PULSE_TIME TCP/IP 設定を使用するべきです。これは、単一の接続だけに対してではなく、所有する TCP/IP パーティションに対してキープアライブの仕組みを管理します。

- SO_REUSEADDR

このオプションは、即時ローカル・アドレス再利用を許可するために使用されます。TCP/IP では常に即時再利用が可能なため、このソケットは互換性のためにのみ提供されています。ソケット再利用を使用不可にする方法はありません。

shutdown()

二重接続の特定の終端をシャットダウンするための shutdown() オプション SHUT_RD および SHUT_WR は、TCP/IP for z/VSE ではサポートされていません。両端をシャットダウンする SHUT_RDWR のみがサポートされています。さらに、他のプラットフォームでは、shutdown() を呼び出した後、ソケット記述子は有効なままですが、TCP/IP for z/VSE は、close() の呼び出しも発行されたかのように動作します。従って、アプリケーションが shutdown() の後で close() を呼び出すと、エラー EBADF になります。互換性のため、TCP/IP の LE ソケット API サポートにより、shutdown() 呼び出しの後の保留中のクローズ要求を記憶し、EBADF エラー・コードは発生しません。ただし、shutdown() の呼び出しと close() の呼び出しとの間に新しい socket() 呼び出しが発行された場合、ソケット記述子は TCP/IP スタックによって既に再使用されている可能性があります。これは、CICS ランタイム環境にも当てはまります。特に、プログラムの制御の外側にある別のトランザクションが、既にソケットを割り振り済みである可能性があります。従って、互換性と移植性のために、shutdown() を使用してソケットをクローズする代わりに、close() を使用することをお勧めします。shutdown() の呼び出しは、一度に使用しないようにします。

socket()

TCP/IP for z/VSE は、AF_INET ドメインにおける TCP 接続および UDP 接続をサポートします。つまり、IPPROTO_TCP および IPPROTO_UDP プロトコルのみがサポートされます。IPPROTO_IP (数値 0) は、特殊な処理を引き起こします。ソケット・タイプに従って、マッチするプロトコルが自動的に選択されます。

- SOCK_DGRAM の場合、プロトコル IPPROTO_UDP が選択されます。
- SOCK_STREAM の場合、プロトコル IPPROTO_TCP が選択されます。

タイプ SOCK_RAW のソケットは TCP/IP for z/VSE ではサポートされていません。

例

以下に示すのは、`setsockopt()` 呼び出しの例です。`getsockopt()` オプション・セットの照会方法の例については、149 ページの『getsockopt() — ソケットに関連したオプションの取得』を参照してください。

```
#include <socket.h>

int rc;
int s;
int option_value;
struct linger l;
:
:
/* I want to linger on close */
l.l_onoff  = 1;
l.l_linger = 100;
rc = setsockopt(s, SOL_SOCKET, SO_LINGER, &l, sizeof(l));
```

shutdown() — 接続のシャットダウン

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

long shutdown(int socket, int how);
```

概要

shutdown() 呼び出しは、接続をシャットダウンします。

パラメーター
　説明

socket　ソケット記述子。

how　シャットダウンの条件。*how* は、以下の値になる可能性があります。

- **SHUT_RD**。*socket* によって指示されたソケットから受信する通信を終了します。
- **SHUT_WR**。*socket* によって指示されたソケットに送信する通信を終了します。
- **SHUT_RDWR**。*socket* によって指示されたソケットに対する送受信両方の通信を終了します。

戻り値

値 0 は成功を表します。値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
　説明

EBADF
　socket が無効ソケット記述子です。

EINVAL
　how パラメーターが、有効値の 1 つに設定されませんでした。

ENOBUFS
　使用可能なシステム・リソースが不十分で、呼び出しを完了させることができません。

ENOTCONN
　ソケットが接続していません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は ENOTCONN に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

スタックの特性

TCP/IP for z/VSE では、二重接続の特定の終端をシャットダウンするために使用される shutdown() のオプション SHUT_RD および SHUT_WR はサポートされません。両端をシャットダウンする SHUT_RDWR のみがサポートされています。さ

らに、他のプラットフォームでは、shutdown() を呼び出した後、ソケット記述子は有効なままですが、TCP/IP for z/VSE は、close() の呼び出しも発行されたかのように動作します。従って、アプリケーションが shutdown() の後で close() を呼び出すと、エラー EBADF になります。互換性のため、TCP/IP の LE ソケット API サポートにより、shutdown() 呼び出しの後の保留中のクローズ要求を記憶し、EBADF エラー・コードは発生しません。ただし、shutdown() の呼び出しと close() の呼び出しとの間に新しい socket() 呼び出しが発行された場合、ソケット記述子は TCP/IP スタックによって既に再使用されている可能性があります。これは、CICS ランタイム環境にも当てはまります。特に、プログラムの制御の外側にある別のトランザクションが、既にソケットを割り振り済みである可能性があります。従って、互換性と移植性のために、shutdown() を使用してソケットをクローズする代わりに、close() を使用することをお勧めします。shutdown() の呼び出しは、一度に使用しないようにします。

socket()

TCP/IP for z/VSE は、AF_INET ドメインにおける TCP 接続および UDP 接続をサポートします。つまり、IPPROTO_TCP および IPPROTO_UDP プロトコルのみがサポートされます。IPPROTO_IP (数値 0) は、特殊な処理を引き起こします。ソケット・タイプに従って、マッチするプロトコルが自動的に選択されます。

- SOCK_DGRAM の場合、プロトコル IPPROTO_UDP が選択されます。
- SOCK_STREAM の場合、プロトコル IPPROTO_TCP が選択されます。

タイプ SOCK_RAW のソケットは TCP/IP for z/VSE ではサポートされていません。

socket() — ソケットの作成

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>
#include <in.h>

int socket(int domain, int type, int protocol);
```

概要

socket() 呼び出しは、通信の端点を作成し、その端点を表すソケット記述子を戻します。異なるタイプのソケットで、異なる通信サービスが提供されます。

パラメーター
説明

domain
要求されたアドレス・ドメイン (AF_INET または AF_INET6)。

type 作成するソケットのタイプ。SOCK_STREAM または SOCK_DGRAM のどちらかです。

protocol
要求されるプロトコル。可能な値は、0、IPPROTO_UDP、または IPPROTO_TCP です。

domain パラメーターによって、通信が行われる通信ドメインが指定されます。このパラメーターにより、使用するアドレス・ファミリー (ドメイン内のアドレスの形式) が選択されます。サポートされるファミリーは、インターネット・ドメインである AF_INET または AF_INET6 です。この定数は、socket.h インクルード・ファイル内に定義されています。

type パラメーターによって、作成されるソケットのタイプが指定されます。タイプは、要求される通信のセマンティクスと類似しています。これらのソケット・タイプ定数は、socket.h インクルード・ファイル内に定義されています。サポートされるパラメーターは以下のとおりです。

ソケット・タイプ
: 説明

SOCK_DGRAM
: その信頼性が保証されていない、固定最大長のコネクションレス・メッセージであるデータグラムを提供します。データグラムでは、破壊、順序が狂った受信、紛失、または複数回の送達が起こる場合があります。このタイプは、AF_INET または AF_INET6 ドメインでサポートされています。

SOCK_STREAM
: 信頼性があり、コネクション指向の、順次両方向バイト・ストリームを提供します。アウト・オブ・バンドのデータのメカニズムがサポートされます。このタイプは、AF_INET または AF_INET6 ドメインでサポートされています。

注: ロー・ソケットはサポートされません

socket() パラメーターについて

protocol パラメーターによって、ソケットで使用される特定のプロトコルが指定されます。ほとんどの場合、特定のアドレス・ファミリーの特定のタイプのソケットごとに、それをサポートするプロトコルが 1 つあります。*protocol* パラメーターが 0 に設定された場合、要求されたドメインおよびソケット・タイプに対するデフォルトのプロトコル番号がシステムによって選択されます。getprotobyname() 呼び出しを使用すると、名前が分かっているプロトコルのプロトコル番号を取得できます。

SOCK_STREAM ソケットは、二重バイト・ストリームをモデル化したものです。ピア・アプリケーション・プログラム間の、信頼性のあるフロー制御接続を提供します。ストリーム・ソケットは、能動または受動のいずれかです。能動ソケットは、connect() を使用して接続要求を開始するクライアントによって使用されます。デフォルトでは、socket() は、能動ソケットを作成します。受動ソケットは、connect() 呼び出しによる接続要求を受け入れるために、サーバーによって使用されます。bind() 呼び出しを使用してソケットに名前を結合し、listen() 呼び出しを使用して接続を受け入れることを示すことによって、能動ソケットを受動ソケットに変換できます。ソケットが受動になった後には、接続要求を開始するためにそのソケットを使用することはできません。

AF_INET および AF_INET6 ドメインでは、ストリーム・ソケットに bind() 呼び出しを適用することにより、アプリケーション・プログラムで、受け入れる接続要求の発信元ネットワークを指定することができます。address 構造体の *Internet address* フィールドを、ネットワーク・インターフェースの IP アドレスに設定する

ことによって、アプリケーション・プログラムで、ネットワーク・インターフェースを完全に指定することができます。または、アプリケーション・プログラムは、ワイルドカードを使用して、どのネットワークからも接続要求を受信することを指定できます。AF_INET ソケットの場合、これを行うには、address 構造体の中の *Internet address* フィールドを、in.h の定義に従って、定数 INADDR_ANY に設定します。AF_INET6 ソケットの場合、これを行うには、address 構造体の中の *Internet address* フィールドを、in.h の定義に従って、IN6ADDR_ANY に設定します。

ストリーム・ソケット間で接続が確立された後には、データ転送呼び出しをどれでも使用することができます (read()、readv()、recv()、recvfrom()、send()、sendto()、write()、および writev())。通常は、read()-write() または send()-recv() ペアが、ストリーム・ソケットでのデータの送信に使用されます。アウト・オブ・バンドのデータが交換される場合には、通常は send()-recv() ペアが使用されます。

SOCK_DGRAM ソケットは、データグラムをモデル化したものです。信頼性の保証がない、コネクションレス・メッセージ交換を提供します。送信メッセージのサイズは最大です。

データグラム・ソケットでは、ストリーム・ソケットでの能動または受動に類似するものはありません。サーバーは bind() を呼び出してソケットに名前を付け、そこからパケットを受信するネットワーク・インターフェースを指定する必要があります。ストリーム・ソケットについての説明のように、ワイルドカードのアドレッシングは、データグラム・ソケットにも適用されます。データグラム・ソケットはコネクションレスなので、listen() 呼び出しは意味を持たず、使用してはなりません。

アプリケーション・プログラムは、データグラム・ソケットを受け取った後、sendto() および recvfrom() 呼び出し、または sendmsg() および recvmsg() 呼び出しを使用して、データグラムを交換できます。アプリケーション・プログラムがさらに処理を進めて、connect() を呼び出し、すべてのメッセージの交換に使用されるピアの名前を完全に指定すると、その他のデータ転送呼び出しである read()、write()、readv()、 writev()、send()、および recv() も使用できるようになります。ソケットを接続状態にする方法については、 118 ページの『connect() — ソケットの接続』を参照してください。

データグラム・ソケットを使用して、複数の宛先にメッセージをブロードキャストすることができます。宛先アドレスをブロードキャスト・アドレスに設定する方法は、ネットワーク・インターフェースによって異なります。これは、アドレス・クラスと、サブネット (ルーティングを単純化するためにより小さい物理ネットワークに分割された論理ネットワーク) が使用されるかどうかに依存します。

ソケットは close() 呼び出しで割り振り解除されます。

戻り値

ソケット記述子が負でない場合、成功を表します。値 -1 はエラーを表します。エラー・コードの値によって、具体的なエラーを示します。

エラー・コード
: 説明

EAFNOSUPPORT
: アドレス・ファミリーがサポートされていません (これは AF_INET または AF_INET6 ではありません)。

EINVAL
: 要求が無効か、またはサポートされていません。

ENOBUFS
: 使用可能なシステム・リソースが不十分で、呼び出しを完了させることができません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

スタックの特性

TCP/IP for z/VSE は、AF_INET ドメインにおける TCP 接続および UDP 接続をサポートします。IPPROTO_TCP および IPPROTO_UDP プロトコル・オプションのみがサポートされます。IPPROTO_IP (数値 0) は、特殊な処理を引き起こします。ソケット・タイプに従って、マッチするプロトコルが自動的に選択されます。

- SOCK_DGRAM の場合、プロトコル IPPROTO_UDP が選択されます。
- SOCK_STREAM の場合、プロトコル IPPROTO_TCP が選択されます。

タイプ SOCK_RAW のソケットは TCP/IP for z/VSE ではサポートされていません。

例

socket() 呼び出しの例を以下に示します。

```
int s;
char *name;
⋮
/* Get stream socket in Internet domain with default protocol */
s = socket (AF_INET, SOCK_STREAM, 0);
⋮
```

socketpair() — ソケットの対の作成

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

int socketpair(int *domain, int type, int protocol, int sv[2]);
```

概要

socketpair() 呼び出しによって、指定済みタイプのソケットの対が取得されますが、これらは名前がなく、指定ドメインで接続されており、指定プロトコルを使用しています。

パラメーター
　　説明

domain
　　ソケットをオープンするドメイン。

type　作成されるソケットのタイプ。

protocol
　　要求されるプロトコル。

sv　取得されたソケットを参照するのに使用される記述子です。

戻り値

ソケット記述子が負でない場合、成功を表します。値 -1 はエラーを表します。具体的なエラーを errno の値が示します。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

takesocket() — 別プログラムからのソケットの取得

フォーマット

```
#define _OPEN_SYS_SOCK_EXT
#include <types.h>
#include <socket.h>

int takesocket(struct clientid *clientid,int sdesc);
```

概要

takesocket() 呼び出しは他のプログラムからソケットを取得します。一般的には、プログラムは、そのスタートアップ・パラメーター・リストを介して、他のプログラムのクライアント ID およびソケット記述子を渡されます。

パラメーター
　　説明

clientid
　　ソケットを取得する元のアプリケーションの *clientid* へのポインター。

sdesc　取得されるソケットの記述子。

clientid 構造体の *c_reserved.type* フィールドが givesocket() 呼び出しで SO_CLOSE に設定された場合は、takesocket() に対する入力として、*clientid* 構造体の *c_close.SockToken* を、通常のソケット記述子の代わりに使用する必要があります。*clientid* 構造体の説明については、 154 ページの『givesocket() — 指定したソケットを使用可能にする』を参照してください。

戻り値

値 -1 はエラーを表します。具体的なエラーを errno の値が示します。-1 以外の場合、戻り値は新しいソケット記述子です。

エラー・コード
: 説明

EBADF
: *sdesc* パラメーターが他のアプリケーション所有の有効なソケット記述子を指定していないか、ソケットはすでに取得されました。

EFAULT
: 指定された *clientid* パラメーターをそのまま使用すると、呼び出し元のパーティションの外側にあるストレージにアクセスする結果になります。

EINVAL
: *clientid* パラメーターが有効なクライアント ID を指定していません。クライアント・プロセスを見つけることができないか、あるいはクライアントが存在していますが、未解決の givesocket がありません。

ENFILE
: ソケット記述子テーブルは、すでにフルになっています。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

termapi() — サブタスク用ソケット API の終了

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <socket.h>

void termapi(void);
```

概要

termapi() 関数は、サブタスク用のソケット API を終了します。

戻り値

TCP/IP 製品が何もインストールされていない場合、または TCP/IP 製品がこの特定の関数を実装していない場合は、メッセージ EDCV001I または EDCT002I が出されます。

write() — ソケットでのデータ書き込み

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <unistd.h>

ssize_t write(int fs, const void *buf, ssize_t N);
```

概要

write()呼び出しは、記述子 *fs* のソケットでバッファーからデータを書き込みます。write() 呼び出しは、接続されたソケットでのみ使用できます。この呼び出しは、最大で N バイトのデータを書き込みます。

write() は、フラグが設定されていない **send()** と同じです。

パラメーター
: 説明

socket
: ソケット記述子。

buf
: 書き込まれるデータが入っているバッファーを指すポインター。

N
: *buf* パラメーターが指すバッファーの長さ (バイト単位)。APAR PQ55591 の PTF がインストールされていない場合、指定される最大バイト数は 64 K です。

送信されるソケット・データを収容するのに使用可能なバッファー・スペースが十分でなく、ソケットがブロッキング・モードの場合、write() は追加バッファー・スペースが使用可能になるまで呼び出し元をブロックします。ソケットが非ブロッキング・モードの場合、write() は -1 を戻し、エラー・コードを EWOULDBLOCK に設定します。非ブロッキング・モードの設定方法の説明については、122 ページの『fcntl() — オープンされたソケット記述子の制御』または 180 ページの『ioctl() — ソケットの制御』を参照してください。

ソケットがデータを受け入れる準備ができていない場合、プロセスがソケットにデータを書き込もうとすると、以下のようになります。

- O_NDELAY が設定されていない限り、write() は、ソケットでデータの受け入れが可能になるまでブロックします。
- O_NDELAY が設定されると、write() は 0 を戻します。

データグラム・ソケットの場合、データグラムが TCP/IP バッファーに適合すれば、この呼び出しはデータグラム全体を送信します。ストリーム・ソケットは、データを分離する境界のない情報ストリームのように動作します。例えば、アプリケーション・プログラムで 1000 バイトを送信する必要がある場合、この関数へのそれぞれの呼び出しによって、1 バイト、10 バイト、または 1000 バイト全体を送信できます。したがって、ストリーム・ソケットを使用するアプリケーション・プログラムはこの呼び出しをループに配置して、すべてのデータが送信されるまでこの関数を呼び出す必要があります。

戻り値

write() は、成功した場合、*N* 以下の実際に書き込まれたバイト数を戻します。成功しなかった場合は、-1 を戻し、errno を以下のいずれかに設定します。

0 またはそれより大きい値は、送信されたバイト数を示しています。しかし、これはデータの送達が完了したことを保証するわけではありません。

EBADF
: *fs* が無効ソケット記述子です。

ECONNRESET
接続はピアによって強制的にクローズされました。

EDESTADDRREQ
ソケットはコネクション指向でなく、ピア・アドレスが設定されていません。

EFAULT
buf および *N* パラメーターを使用すると、呼び出し元アドレス・スペースの外側のストレージへのアクセスを試みる結果になります。

EINVAL
要求が無効か、またはサポートされていません。

EIO 入出力エラーが発生しました。

ENOBUFS
メッセージの送信にバッファー・スペースを使用することができません。

ENOTCONN
ソケットが接続していません。

EWOULDBLOCK
ソケットは非ブロッキング・モードであり、データは書き込みに使用可能でありません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

例

以下に示すのは、write() 呼び出しの例です。

```
#include <stdio.h>
#include <string.h>

/*Write the zero terminated string aString to the socket aSocket and
print number of bytes written. Return number of bytes written or -1
for no success.
*/
int writeToSocket(int aSocket, char* aString)
{ int numberOfBytesWritten;

 number ofBytesWritten=
   write(aSocket, aString, strlen(aString));
 if (numberOfBytesWritten < 0)
 { perror("write"); return -1; }
 else
 { printf("number of bytes written is %d.¥n", numberOfBytesWritten);
   return numberOfBytesWritten:
 }
}
```

writev() — 配列からソケットへのデータの書き込み

フォーマット

```
#define _XOPEN_SOURCE_EXTENDED 1
#include <uio.h>

ssize_t writev(int fs, const struct iovec *iov, int iovcnt);
```

概要

writev() 呼び出しは、一組のバッファーから、記述子 *fs* をもつソケットに、データを書き込みます。データは、*iov[0]...iov[iovcnt-1]* で指定されたバッファーから収集されます。記述子は、接続先のソケットを表している必要があります。

パラメーター
　説明

fs　ソケット記述子。

iov　**iovec** バッファーの配列を指すポインター。

iovcnt　*iov* パラメーターが指すバッファー数。

iovec 構造体は **uio.h** で定義されており、以下のフィールドが含まれています。

エレメント
　説明

iov_base
　バッファーを指すポインター。

iov_length
　バッファーの長さ。

この呼び出しは、データの *iov_length* バイトの合計を書き込みます。

送信されるソケット・データを収容するのに使用可能なバッファー・スペースが十分でなく、ソケットがブロッキング・モードの場合、writev() は追加バッファー・スペースが使用可能になるまで呼び出し元をブロックします。ソケットが非ブロッキング・モードの場合、writev() は -1 を戻し、エラー・コードを EWOULDBLOCK に設定します。

戻り値

正常に実行された場合は、バッファーから書き込まれたバイト数が返されます。値 -1 はエラーを表します。具体的なエラーを errno の値が示します。

0 またはそれより大きい値は、送信されたバイト数を示していますが、これはデータの送達が完了したことを保証するわけではありません。

TCP/IP 製品が何もインストールされていない場合、または、TCP/IP 製品がこの特定の関数をインプリメントしていない場合、C ランタイムの対応するダミー・ルーチンは常に値 -1 を戻し、errno は EINVAL に設定されます。この場合、メッセージ EDCV001I または EDCT002I が出されます。

第 12 章 CALL 命令アプリケーション・プログラム・インターフェース (EZASOKET API) の使用

このトピックでは、TCP/IP アプリケーション・プログラム用の CALL 命令 API について説明します。本章には、以下のトピックがあります。

- 環境に関する制約事項およびプログラミング要件
- CALL 命令 API
- COBOL、アセンブラー、および PL/I の呼び出しフォーマット

環境に関する制約事項およびプログラミング要件

呼び出し可能ソケット API には、以下の制約事項が適用されます。

- EZASOKET API は、ICCF 疑似パーティションで実行しているプログラムで使用することはできません。
- ロック

  これらの呼び出しを発行するときには、ロックが保持されているべきではありません。
- INITAPI/TERMAPI マクロ

  INITAPI/TERMAPI マクロは、同じタスク下で発行されなければなりません。
- ストレージ

  ソケット呼び出しから戻されるデータを収容する目的のために獲得するストレージは、ソケット呼び出し時のアプリケーション・プログラム状況ワード (PSW) と同じキーで取得されなければなりません。
- アドレス指定モード (AMODE) の考慮事項

  EZASOKET CALL API は、呼び出し元が 31 ビット AMODE である間に起動されなければなりません。
- CICS がストレージ保護で動作しているときに CICS トランザクション内で EZASOKET CALL API を使用する場合、この CALL API を使用するすべてのプログラムは、EXECKEY(CICS) で定義されている必要があります。それらのプログラムにリンクされているプログラムについても同様です。トランザクション定義の TASKDATAKEY(CICS) は不要です。
- CICS トランザクション内で CALL API を使用する場合、EZA 「タスク関連ユーザー出口」(TRUE) が活動化された後でないと、これらのトランザクションは実行できません。この TRUE を活動化する方法について詳しくは、 93 ページの『EZA インターフェースに関する CICS の考慮事項』を参照してください。
- LE/C ソケット API マルチプレクサーを使用して、アプリケーションで使用する TCP/IP および SSL の実装を選択する方法については、 95 ページの『第 10 章 使用する TCP/IP と SSL の実装を選択』を参照してください。

CALL 命令アプリケーション・プログラム・インターフェース (API)

このセクションでは、COBOL、PL/I、または High Level Assembler で作成される TCP/IP アプリケーション・プログラム用の CALL 命令 API について説明します。各ソケット呼び出しごとに、フォーマットとパラメーターを説明します。

注:

1. このインターフェースでは再入可能コードがサポートされます。
2. PL/I プログラムの場合、最初の呼び出し命令の前に、次のステートメントを組み込んでください。

   ```
   DCL EZASOKET ENTRY OPTIONS(RETCODE,ASM,INTER) EXT;
   ```

3. レジスター規則:

   レジスター 0、1、14、および 15 は、インターフェースによって使用されるので、必要であれば起動前に保管が必要です。

   レジスター 13 は、呼び出し元が用意する 72 バイトの保管域を示さなければなりません。

COBOL、アセンブラー、および PL/I の呼び出しフォーマット

この API は、EZASOKET プログラムを呼び出すことで起動され、C 言語呼び出しと同じ機能を実行します。プログラミング言語の違いのため、パラメーターは異なります。

COBOL 言語の呼び出しフォーマット

```
►►—CALL 'EZASOKET' USING SOC-FUNCTION—parm1 parm2 ...—ERRNO RETCODE.————————►◄
```

SOC-FUNCTION
16 バイトの文字フィールドであり、左寄せにして、右にブランクが埋められます。呼び出しの名前に設定します。SOC-FUNCTION は、大/小文字が決まっています。大文字でなければなりません。

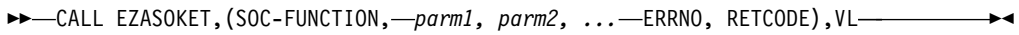
parm*n*
パラメーター数は、呼び出しのタイプによって可変です。

ERRNO
RETCODE が負の場合、ERRNO にエラー番号が入ります。このフィールドは、呼び出しのすべてではありませんが、ほとんどで使用されます。

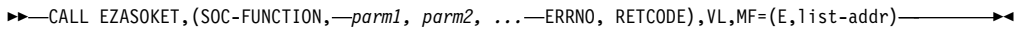
RETCODE
EZASOKET 呼び出しから戻されるコードが入るフルワード 2 進数の変数です。この値は、C 関数の通常の戻り値に対応します。

アセンブラー言語の呼び出しフォーマット

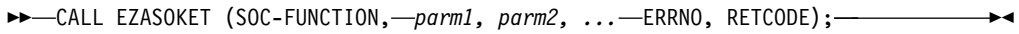
アセンブラー言語プログラムの 'EZASOKET' 呼び出しフォーマットは、次のとおりです。



再入可能プログラミングには、以下の呼び出しフォーマットを使用できます。



PL/I 言語の呼び出しフォーマット



SOC-FUNCTION
16 バイトの文字フィールドであり、左寄せにして、右にブランクが埋められます。呼び出しの名前に設定します。

parm*n*
パラメーター数は、呼び出しのタイプによって可変です。

ERRNO
RETCODE が負の場合、ERRNO にエラー番号が入ります。このフィールドは、呼び出しのすべてではありませんが、ほとんどで使用されます。

RETCODE
EZASOKET 呼び出しから戻されるコードが入るフルワード 2 進数の変数です。この値は、C 関数の通常の戻り値に対応します。

パラメーター記述の変換

このトピックのパラメーター記述は、COBOL VSE 言語の構文と規則を使用して書かれていますが、使用する言語に適した構文と規則を使用する必要があります。

232 ページの図 16 に、COBOL、PL/I、およびアセンブラーの各言語プログラムでのストレージ定義ステートメントの例を示します。

```
COBOL PIC


  PIC S9(4) COMP                    HALFWORD BINARY VALUE
  PIC S9(8) COMP                    FULLWORD BINARY VALUE
  PIC   X(n)                        CHARACTER FIELD OF N BYTES

PL/I DECLARE STATEMENT


  DCL   HALF       FIXED BIN(15),   HALFWORD BINARY VALUE
  DCL   FULL       FIXED BIN(31),   FULLWORD BINARY VALUE
  DCL   CHARACTER  CHAR(n)          CHARACTER FIELD OF n BYTES

ASSEMBLER DECLARATION


  DS    H                           HALFWORD BINARY VALUE
  DS    F                           FULLWORD BINARY VALUE
  DS    CLn                         CHARACTER FIELD OF n BYTES
```

図 *16.* ストレージ定義ステートメントの例

エラー・メッセージおよび戻りコード

エラー・メッセージについては、「*IBM z/VSE* メッセージおよびコード 第 *1* 巻」および「*TCP/IP for VSE Messages*」を参照してください。

TCP/IP によって戻されるエラー・コードについては、 83 ページの『ERRNO 値』を参照してください。

デバッグ

615 ページの『付録 B. EZASMI および EZASOKET インターフェースのデバッグ機能 (EZAAPI トレース)』を参照してください。

CALL 命令のコーディング

このセクションでは、この API に含まれる各呼び出し命令について、説明、構文、パラメーター、およびその他の関連情報を説明します。

ACCEPT

サーバーは、クライアントからの接続要求を受け入れるために、ACCEPT 呼び出しを発行します。

この呼び出しは、前もって SOCKET 呼び出しによって作成されて LISTEN 呼び出しによってマークされたソケットを指します。

ACCEPT 呼び出しは、ブロッキング呼び出しです。発行されると、ACCEPT 呼び出しは次のことを行います。

1. 保留中接続の待ち行列の最初の接続を受け入れます。

2. S と同じプロパティーを持つ新規ソケットを作成し、その記述子を RETCODE に戻します。元のソケットを、呼び出し側プログラムがさらに接続要求を受け入れるために使用し続けることができます。
3. クライアントのアドレスが、後続のサーバー呼び出しで使用できるように NAME に戻されます。

注:

1. ソケットがブロッキング・モードまたは非ブロッキング・モードのどちらのモードなのかは、特定の各コマンドの操作に影響します。デフォルトはブロッキングです。非ブロッキング・モードは、FCNTL および IOCTL 呼び出しの使用によって確立できます。ソケットがブロッキング・モードである場合、入出力呼び出しは、特定の各イベントの完了を待機します。例えば、READ 呼び出しは、バッファーに入力データが収容されるまでブロックします。入出力呼び出しが発行された場合に、ソケットがブロッキングであれば、イベントが完了するまでプログラム処理は中断されます。ソケットが非ブロッキングであれば、プログラム処理は継続します。
2. 保留中の接続要求が待ち行列にない場合、ACCEPT は、ソケットが非ブロッキング・モードでなければ、ソケットをブロックします。FCNTL または IOCTL を呼び出すことによって、ソケットを非ブロッキングに設定できます。
3. 複数のソケット呼び出しが発行される場合、ACCEPT の前に SELECT 呼び出しを発行して、保留中の接続要求があることを確認できます。この技法では、後続の ACCEPT 呼び出しはブロックしません。
4. TCP/IP には、クライアントを選別する機能は備わっていません。どの接続要求を受け入れるのかを制御するのはアプリケーション・プログラムに任されますが、クライアントの ID を検出した後ですぐに接続をクローズすることはできます。

重要: IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'ACCEPT'.
    01  S               PIC 9(4) BINARY.
    01  NAME.
        03  FAMILY      PIC 9(4) BINARY.
        03  PORT        PIC 9(4) BINARY.
        03  IP-ADDRESS  PIC 9(8) BINARY.
        03  RESERVED    PIC X(8).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'ACCEPT' を含みます。このフィールドは、左寄せにして右にブランクを埋め込みます。

S ハーフワード 2 進数であり、前に SOCKET 呼び出しで作成されたソケットの記述子を指定します。並行サーバーでは、これは、サーバーがそこで listen を行うソケットです。

アプリケーションへ戻されるパラメーター値

NAME
最初は、IPv4 または IPv6 アプリケーションが IPv4 または IPv6 ソケット・アドレス構造体へのポインターを設定します。呼び出しの完了時には、その構造体には、接続ピアのソケット・アドレスが挿入されます。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
説明

FAMILY
ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、クライアントのポート番号を指定します。

IPv4-ADDRESS
フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
説明

NAMELEN
IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、クライアントのポート番号に設定されます。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
RETCODE 値が正の場合、その RETCODE 値は新規ソケット番号です。

RETCODE 値が負の場合、ERRNO フィールドのエラー番号をチェックしてください。

BIND

典型的なサーバー・プログラムでは、BIND 呼び出しは SOCKET 呼び出しの後に続き、新規ソケットの作成プロセスを完了します。

BIND 呼び出しでは、必要なポートを指定したり、システムによるポートの選択を許可したりできます。リスナー・プログラムを常に同じウェルノウン・ポートにバインドする必要があります。このようにすることで、クライアントは、接続を試みる際にどのソケット・アドレスを使用するのかを把握することができます。

AF_INET ドメインでは、ストリーム・ソケットに対する BIND 呼び出しは、どのネットワークから接続要求を受け入れるのかを指定できます。アプリケーションは、ADDRESS フィールドをネットワーク・インターフェースの IP アドレスに設定することによって、ネットワーク・インターフェースを完全に指定できます。または、アプリケーションは、ワイルドカード を使用して、どのネットワーク・インターフェースからでも接続要求を受け取ることを指定できます。これは、ADDRESS フィールドをフルワードのゼロに設定することで行えます。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'BIND'.
    01  S               PIC 9(4) BINARY.
    01  NAME.
        03  FAMILY      PIC 9(4) BINARY.
        03  PORT        PIC 9(4) BINARY.
        03  IP-ADDRESS  PIC 9(8) BINARY.
        03  RESERVED    PIC X(8).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、BIND を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S ハーフワード 2 進数であり、バインドされるソケットのソケット記述子を指定します。

NAME

IPv4 または IPv6 アプリケーションは、IPv4 または IPv6 ソケット・アドレス構造体へのポインターを設定します。この構造体は、アプリケーションが接続を受け入れることができるポート番号と、IPv4 または IPv6 IP アドレスを指定します。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
: 説明

FAMILY
: ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
: ハーフワード 2 進数フィールドであり、クライアントのポート番号を指定します。ポート番号をゼロに設定すると、TCP/IP がポート

を割り当てます。アプリケーションは、BIND マクロの後に GETSOCKNAME マクロを呼び出して、割り当てられたポートを知ることができます。

IPv4-ADDRESS
フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
説明

NAMELEN
IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、クライアントのポート番号に設定されます。ポート番号をゼロに設定すると、TCP/IP がポートを割り当てます。アプリケーションは、BIND マクロの後に GETSOCKNAME マクロを呼び出して、割り当てられたポートを知ることができます。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

アプリケーションへ戻されるパラメーター値

ERRNO

フルワード 2 進数フィールド。RETCODE が負の値の場合、このフィールドの中にはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

CLOSE

CLOSE 呼び出しは、以下の機能を実行します。

- CLOSE 呼び出しは、ソケットをシャットダウンし、そのソケットに割り振られているすべてのリソースを解放します。ソケットが参照する TCP 接続がオープンしている場合、その接続はクローズされます。
- CLOSE 呼び出しは、並行サーバーによっても発行されます。その場合、サーバーが子サーバー・プログラムにソケットを与えた後に発行されます。並行サーバーは、GIVESOCKET を発行し、子クライアントが正常に TAKESOCKET を発行したという通知を受け取った後、close コマンドを発行して所有権の引き渡しを完了します。トランザクション・ベースのハイパフォーマンスのシステムでは、CLOSE 呼び出しに関連するタイムアウトによってパフォーマンス上の問題が起こることがあります。

  注:

  1. 待ち行列に入力データまたは出力データがあるときにストリーム・ソケットがクローズされると、TCP 接続はリセットされ、データ伝送が不完全になることがあります。SETSOCKOPT 呼び出しを使用してリンガー 条件を設定することができ、そうすると、TCP/IP は、CLOSE 呼び出しが発行された後、指定された期間は引き続きデータ伝送を完了しようとします。 311 ページの『SETSOCKOPT』の説明にある SO-LINGER を参照してください。
  2. 並行サーバーは、反復サーバーとは違います。反復サーバーは、ある 1 時点で 1 つのクライアントにサービスを提供します。並行サーバーは、複数のクライアントから接続要求を受け取り、それらのクライアントに実際にサービスを提供する複数の子サーバーを作成します。子サーバーが作成されると、並行サーバーは新しいソケットを取得し、その新規ソケットを子サーバーに渡すと、自身を接続から切り離します。CICS リスナーは、並行サーバーの 1 つの例です。
  3. 成功しなかったソケット呼び出しの後には、close の発行と、新規ソケットのオープンが実行されるべきです。同じソケットを別の呼び出しで使用すると、戻りコードがゼロ以外の値になります。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'CLOSE'.
    01  S               PIC 9(4) BINARY.
```

```
01  ERRNO          PIC 9(8) BINARY.
01  RETCODE        PIC S9(8) BINARY.


CALL 'EZASOKET' USING SOC-FUNCTION S ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、CLOSE を含みます。このフィールドは、左寄せにして右にブランクを埋め込みます。

S
ハーフワード 2 進数フィールドであり、クローズされるソケットの記述子を含みます。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の値の場合、このフィールドの中にはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

CONNECT

CONNECT 呼び出しは、ローカル・ソケットとリモート・ソケットの間に接続を確立するためにクライアントが発行します。

ストリーム・ソケット

ストリーム・ソケットの場合、CONNECT 呼び出しは、サーバーとの接続を確立するために、クライアントによって発行されます。この呼び出しは、以下の 2 つのタスクを実行します。

1. 以前に BIND 呼び出しが発行されていない場合、ストリーム・ソケットのバインド処理を完了します。
2. リモート・ソケットへの接続を試みます。データが転送されるには、その前にこの接続が必要です。

UDP ソケット

UDP ソケットの場合、入出力呼び出しの前に CONNECT 呼び出しは必要はありませんが、発行された場合、宛先を指定せずにメッセージを送信できます。

ストリーム・ソケットに対してクライアントおよびサーバーが発行する呼び出しシーケンスは、次のとおりです。

1. サーバー は、BIND および LISTEN を発行して、受動オープン・ソケットを作成します。
2. クライアント は、CONNECT を発行して、接続を要求します。
3. サーバー は、受動オープン・ソケットで接続を受け入れて、接続された新規ソケットを作成します。

CONNECT 呼び出しのブロッキング・モードが、呼び出しの動作に影響します。

- ソケットがブロッキング・モードの場合、CONNECT 呼び出しは、接続が確立するまで、または、エラーを受け取るまで、呼び出し側プログラムをブロックします。
- ソケットが非ブロッキング・モードの場合、接続要求が成功したかどうかを戻りコードが示します。
  - ゼロの RETCODE は、接続が完了したことを示します。
  - RETCODE がゼロ以外で、ERRNO が EINPROGRESS の場合、接続が完了していないことを示しますが、ソケットが非ブロッキングであるため CONNECT 呼び出しは正常に戻ります。

  呼び出し元は、SELECT の呼び出しと、ソケットへの書き込みテストを行うことによって、接続セットアップの完了をテストする必要があります。

完了は、2 回目の CONNECT を発行することではチェックできません。詳しくは、 300 ページの『SELECT』を参照してください。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'CONNECT'.
    01  S               PIC 9(4) BINARY.
    01  NAME.
        03  FAMILY      PIC 9(4) BINARY.
        03  PORT        PIC 9(4) BINARY.
        03  IP-ADDRESS  PIC 9(8) BINARY.
        03  RESERVED    PIC X(8).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.


CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、CONNECT を含みます。このフィールドは、左寄せにして右にブランクを埋め込みます。

S
ハーフワード 2 進数であり、接続を確立するために使用されるソケットのソケット記述子を指定します。

NAME

入力パラメーター。CONNECT の NAME パラメーターは、ローカルのクライアント・ソケットが接続される、ターゲットの IPv4 または IPv6 ソケット・アドレスを指定します。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
: 説明

FAMILY
: ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
: ハーフワード 2 進数フィールドであり、クライアントのポート番号を指定します。

IPv4-ADDRESS
: フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
: 2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
: 説明

NAMELEN
: IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
: IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
: ハーフワード 2 進数フィールドであり、クライアントのポート番号に設定されます。

FLOW-INFO
: フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の値の場合、このフィールドの中にはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

FCNTL

ソケットのブロッキング・モードは、FCNTL 呼び出しに記述される FNDELAY フラグを使用して、照会するか、または非ブロッキングに設定することができます。

FNDELAY フラグは、プログラム内に定義されていなくても、照会または設定することができます。

ソケットのブロッキング・モードを制御する別の方法については、 287 ページの『IOCTL』を参照してください。

以下の例は、FCNTL 呼び出し命令を示しています。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'FCNTL'.
    01  S               PIC 9(4) BINARY.
    01  COMMAND         PIC 9(8) BINARY.
    01  REQARG          PIC 9(8) BINARY.
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.


PROCEDURE
CALL 'EZASOKET' USING SOC-FUNCTION S COMMAND REQARG
                ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、FCNTL を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S
ハーフワード 2 進数であり、非ブロッキングに設定するか照会するソケットの、ソケット記述子を指定します。

COMMAND
フルワード 2 進数であり、以下の値を指定します。

値	説明
3	ソケットのブロッキング・モードを照会する
4	ソケットのモードをブロッキングまたは非ブロッキングに設定する

REQARG
フルワード 2 進数フィールドであり、TCP/IP が FNDELAY フラグを設定するのに使用するマスクを含みます。

- COMMAND が 3 (照会) に設定されている場合、REQARG フィールドは 0 に設定される必要があります。
- COMMAND が 4 (設定) に設定されている場合は、次のとおりです。
  - REQARG を 4 に設定すると、FNDELAY フラグはオンになります。これにより、ソケットは非ブロッキング・モードになります。
  - REQARG を 0 に設定すると、FNDELAY フラグはオフになります。これにより、ソケットはブロッキング・モードになります。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

- COMMAND が 3 (照会) に設定された場合、ビット・ストリングが戻されます。
  - RETCODE に X'00000004' が入っている場合、ソケットは非ブロッキングです。(FNDELAY フラグはオンです)
  - RETCODE に X'00000000' が入っている場合、ソケットはブロッキングです。(FNDELAY フラグはオフです)
- COMMAND が 4 (設定) に設定された場合、呼び出しが成功したことがこのフィールド内の 0 によって示されます。両方のケースとも、RETCODE -1 はエラーを示します (ERRNO フィールドのエラー番号をチェックしてください)。

FREEADDRINFO

FREEADDRINFO 呼び出しは、RES パラメーターで GETADDRINFO により返されるすべてのアドレス情報構造を解放します。

重要: この関数呼び出しは TCP/IP for z/VSE では使用できません。

以下の例は、FREEADDRINFO 呼び出し命令を示しています。

COBOL の例

```
WORKING-STORAGE SECTION.
    01 SOC-FUNCTION PIC X(16) VALUE IS 'FREEADDRINFO'.
    01 ADDRINFO PIC 9(8) BINARY.
    01 ERRNO PIC 9(8) BINARY.
    01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION ADDRINFO ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、FREEADDRINFO を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

ADDRINFO

GETADDRINFO RES 引数によって返されるアドレス情報構造体セットのアドレス。

アプリケーションへ戻されるパラメーター値

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO フィールドは無視できます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

GETADDRINFO

GETADDRINFO 呼び出しは、サービス・ロケーションの名前 (例えば、ホスト名)、サービス名、またはその両方を、ソケット・アドレスのセットおよび他の関連情報に変換します。

この情報を使用して、ソケットを作成したり、指定したサービスに接続したり、またはそのサービスにデータグラムを送信したりできます。

重要: この関数呼び出しは TCP/IP for z/VSE では使用できません。

COBOL の例

```
WORKING-STORAGE SECTION.
    01 SOC-FUNCTION PIC X(16) VALUE IS 'GETADDRINFO'.
    01 NODE PIC X(255).
    01 NODELEN PIC 9(8) BINARY.
    01 SERVIC PIC X(32).
    01 SERVLEN PIC 9(8) BINARY.
    01 AI-PASSIVE PIC 9(8) BINARY VALUE 1.
    01 AI-CANONNAMEOK PIC 9(8) BINARY VALUE 2.
    01 AI-NUMERICHOST PIC 9(8) BINARY VALUE 4.
    01 AI-NUMERICSERV PIC 9(8) BINARY VALUE 8.
    01 AI-V4MAPPED PIC 9(8) BINARY VALUE 16.
    01 AI-ALL PIC 9(8) BINARY VALUE 32.
    01 AI-ADDRCONFIG PIC 9(8) BINARY VALUE 64.
    01 HINTS USAGE IS POINTER.
    01 RES USAGE IS POINTER.
    01 CANNLEN PIC 9(8) BINARY.
    01 ERRNO PIC 9(8) BINARY.
    01 RETCODE PIC S9(8) BINARY.

LINKAGE SECTION.
    01 HINTS-ADDRINFO.
       03 FLAGS PIC 9(8) BINARY.
       03 AF PIC 9(8) BINARY.
       03 SOCTYPE PIC 9(8) BINARY.
       03 PROTO PIC 9(8) BINARY.
       03 FILLER PIC 9(8) BINARY.
       03 FILLER PIC X(4).
       03 FILLER PIC X(4).
       03 FILLER PIC 9(8) BINARY.
       03 FILLER PIC X(4).
       03 FILLER PIC 9(8) BINARY.
       03 FILLER PIC X(4).
       03 FILLER PIC 9(8) BINARY.
    01 RES-ADDRINFO.
       03 FLAGS PIC 9(8) BINARY.
       03 AF PIC 9(8) BINARY.
       03 SOCTYPE PIC 9(8) BINARY.
       03 PROTO PIC 9(8) BINARY.
       03 NAMELEN PIC 9(8) BINARY.
       03 FILLER PIC X(4).
       03 FILLER PIC X(4).
       03 CANONNAME USAGE IS POINTER.
       03 FILLER PIC X(4).
       03 NAME USAGE IS POINTER.
       03 FILLER PIC X(4).
       03 NEXTP USAGE IS POINTER.

PROCEDURE DIVISION.
    MOVE 'www.hostname.com' TO NODE.
    MOVE 16 TO NODELEN.
    MOVE 'TELNET' TO SERVIC.
    MOVE 6 TO SERVLEN.
    SET HINTS TO ADDRESS OF HINTS-ADDRINFO.
    CALL 'EZASOKET' USING SOC-FUNCTION NODE NODELEN SERVIC SERVLEN HINTS RES CANNLEN
    ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、GETADDRINFO を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

NODE

照会されるホスト名が入る、最大で 255 バイト長のストレージ。AI_NUMERICHOST フラグが、HINTS オペランドにより示されるストレージで指定されている場合、NODE には、ネットワーク・バイト・オーダー表示形式の、照会されるホスト IP アドレスが入らなければなりません。これはオプションのフィールドですが、指定した場合は NODELEN もコーディングする必要があります。照会される NODE 名は、NODELEN まで、または最初の 2 進ゼロまでで構成されます。スコープ情報をホスト名に、*node%scope information* という形式を用いて追加できます。結合される情報は、255 バイト以下でなければなりません。

NODELEN

フルワード 2 進数フィールドであり、NODE フィールドに指定されたホスト名の長さに設定されます。余分のブランクを含めることはできません。これはオプションのフィールドですが、指定した場合は NODE もコーディングする必要があります。

SERVIC

照会されるサービス名が入る、最大で 32 バイト長のストレージ。AI_NUMERICSERV フラグが、HINTS オペランドにより示されるストレージで指定されている場合、SERVIC には、表示形式の、照会されるポート番号が入る必要があります。これはオプションのフィールドですが、指定した場合は SERVLEN もコーディングする必要があります。照会される SERVIC 名は、SERVLEN まで、または最初の 2 進ゼロまでで構成されます。

SERVLEN

フルワード 2 進数フィールドであり、SERVIC フィールドに指定されたサービス名の長さに設定されます。余分のブランクを含めることはできません。これはオプションのフィールドですが、指定した場合は SERVIC もコーディングする必要があります。

HINTS

入力パラメーター。HINTS 引数を指定する場合、それには入力値が含まれる addrinfo 構造体のアドレスが入ります。その入力値によって、オプションを指定したり、返される情報を特定のソケット・タイプ、アドレス・ファミリー、またはプロトコルに制限したりすることで、操作を指示できます。HINTS 引数を指定しない場合、返される情報は、FLAGS、SOCTYPE、および PROTO フィールドには値 0、AF フィールドには値 AF_UNSPEC が含まれる構造体を参照した場合と同じものになります。これはオプションのフィールドです。

アドレス情報構造体には、以下のフィールドがあります。

値　　説明

FLAGS

フルワード 2 進数フィールド。ビット単位の値 0、または以下のいずれか 1 つ以上の値が指定されなければなりません。

AI-PASSIVE (X'00000001') または **10** 進数値の **1**。

返される RES により示される NAME に挿入する方法を指定します。

このフラグを指定した場合、返されるアドレス情報は、指定したサービス (例えば、BIND 呼び出し) の着信接続を受け入れるソケットのバインドでの使用に適しています。この場合、NODE 引数が指定されていないと、返される RES により示されるソケット・アドレス構造体の IP アドレス部分は、IPv4 アドレスの場合は INADDR_ANY、IPv6 アドレスの場合は IPv6 未指定アドレス (in6addr_any) に設定されます。

このフラグが設定されていない場合、返されるアドレス情報は、CONNECT 呼び出し (コネクション型プロトコルの場合)、または CONNECT、SENDTO、SENDMSG 呼び出し (コネクションレス・プロトコルの場合) に適します。この場合、NODE 引数が指定されていないと、返される RES により示されるソケット・アドレス構造体の IP アドレス部分は、IPv4 アドレスの場合はデフォルトのループバック・アドレス、IPv6 アドレスの場合はデフォルトのループバック・アドレスに設定されます。

このフラグは、NODE 引数が指定されている場合は無視されます。

AI-CANONNAMEOK (X'00000002') または **10** 進数値の **2**。
このフラグが指定され、かつ NODE 引数が指定されている場合、GETADDRINFO 呼び出しは、NODE 引数に対応する正規名を決定しようとします。

AI-NUMERICHOST (X'00000004') または **10** 進数値の **4**。
このフラグを指定した場合は、NODE 引数は、表示形式の数値ホスト・アドレスでなければなりません。そうしない場合、ホストが見つからない [EAI_NONAME] というエラーが戻されます。

AI-NUMERICSERV (X'00000008') または **10** 進数値の **8**。
このフラグを指定した場合は、SERVIC 引数は、表示形式の数値ポートでなければなりません。そうしない場合、[EAI_NONAME] というエラーが戻されます。

AI-V4MAPPED (X'00000010') または **10** 進数値の **16**。
このフラグを指定した場合に、AF フィールドに値 AF_INET6、または IPv6 がサポートされている場合には値 AF_UNSPEC を指定すると、呼び出し元は IPv4 マップの IPv6 を受け入れます。また、AI-ALL フラグの指定もなく、IPv6 アドレスが検出されない場合は、IPv4 アドレスの照会が実行されます。IPv4 アドレスが検出された場合、それらは IPv4 マップの IPv6 アドレスとして返されます。

AF フィールドに値 AF_INET6 が入っていないか、または AF フィールドに AF_UNSPEC が入っているものの IPv6 がシステムでサポートされていない場合、このフラグは無視されます。

AI-ALL (X'00000020') または **10** 進数値の **32**。
AF フィールドに値 AF_INET6 が入っており、かつ AI-ALL が設定されている場合、さらに AI-V4MAPPED フラグを、すべてのアドレス (IPv6 および IPv4 マップの IPv6 アドレス) の受け入れを呼び出し元に指示するように設定する必要があります。システムが IPv6 をサポートする場合に AF フィールドに値 AF_UNSPEC が入っており、かつ AI-ALL が設定されている場合、呼び出し元は、IPv6 アドレスと、IPv4 アドレス (AI-V4MAPPED が設定されていない場合) または IPv4 マップの IPv6 アドレス (AI-V4MAPPED が設定されている場合) のいずれかを受け入れます。まず IPv6 アドレスについて照会が行われ、それが正常に実行されると、IPv6 アドレスが返されます。次いで別の照会が IPv4 アドレスについて行われ、検出されたすべての IPv4 アドレスも、IPv4 マップの IPv6 アドレス (AI-V4MAPPED も指定されている場合) または IPv4 アドレス (AI-V4MAPPED が指定されていない場合) のいずれかとして返されます。

AF フィールドに値 AF_INET6 が入っていないか、またはシステムが IPv6 をサポートしているが値 AF_UNSPEC が入っていない場合、このフラグは無視されます。

AI-ADDRCONFIG (X'00000040') または **10** 進数値の **64**。
このフラグが指定されている場合、リゾルバーが以下のいずれかが真であると判断すると、NODE にある名前に対する照会が行われます。

- システムが IPv6 対応であり、少なくとも 1 つの IPv6 インターフェースがある場合、リゾルバーは IPv6 (AAAA または A6 DNS) レコードの照会を行います。
- システムが IPv4 対応であり、少なくとも 1 つの IPv4 インターフェースがある場合、リゾルバーは IPv4 IPv4 (A DNS) レコードの照会を行います。ループバック・アドレスは、この場合には有効なインターフェースとは見なされません。

注: 上記のフラグのバイナリー OR を COBOL プログラムで実行する場合は、以下の例で示すとおり、必要な COBOL ステートメントを追加するだけで済みます。COBOL ADD の後の FLAGS フィールドの値が 10 進数 80 または X'00000050' であることに注意してください。これは AI_V4MAPPED および AI_ADDRCONFIG、または X'00000010' および X'00000040' での OR 実行の合計です。

```
01 AI-V4MAPPED PIC 9(8) BINARY VALUE 16.
01 AI-ADDRCONFIG PIC 9(8) BINARY VALUE 64.

ADD AI-V4MAPPED TO FLAGS.
ADD AI-ADDRCONFG TO FLAGS.
```

AF フルワード 2 進数フィールド。返される情報を特定のアドレス・フ

ファミリーに制限するために使用します。値 AF_UNSPEC は、呼び出し元がどのプロトコル・ファミリーでも受け入れることを意味します。10 進数の値 0 は AF_UNSPEC を示します。10 進数の値 2 は AF_INET を示し、10 進数の値 19 は AF_INET6 を示します。

SOCTYPE

フルワード 2 進数フィールド。返される情報を特定のソケット・タイプに制限するために使用します。値 0 は、呼び出し元がどのソケット・タイプでも受け入れることを意味します。特定のソケット・タイプを指定しない場合 (例えば、値 0)、サポートされるすべてのソケット・タイプに関する情報が返されます。以下に示すのは、受け入れ可能なソケット・タイプです。

タイプ名	**10** 進数値	説明
SOCK_STREAM	1	ストリーム・ソケット用
SOCK_DGRAM	2	データグラム・ソケット用
SOCK_RAW	3	ロー・プロトコル・インターフェース用

これ以外の値では失敗し、戻りコード EAI_SOCTYPE が出されます。SOCK_RAW は受け入れられますが、SERVIC が数値 (例えば、SERVIC=23) の場合にのみ有効であることに注意してください。SERVIC 名の検索は、SOCK_STREAM または SOCK_DGRAM 以外のいずれかのプロトコル値を使用した適切なサービス・ファイルでは、決して行われません。

PROTO が 0 ではなく、かつ SOCTYPE が 0 である場合、PROTO の受け入れ可能な入力値は、IPPROTO_TCP および IPPROTO_UDP のみです。それ以外の場合、GETADDRINFO 呼び出しは失敗し、戻りコード EAI_BADFLAGS が出されます。

SOCTYPE および PROTO の両方が 0 と指定されている場合、GETADDRINFO は以下のように進みます。

- SERVIC が NULL の場合、または SERVIC が数値の場合、返されるアドレス情報により、SOCTYPE の指定はデフォルトの SOCK_STREAM になります。
- SERVIC がサービス名として指定されている場合 (例えば、SERVIC=FTP)、GETADDRINFO 呼び出しは適切なサービス・ファイルを 2 回検索します。最初の検索では SOCK_STREAM がプロトコルとして使用され、2 番目の検索では SOCK_DGRAM がプロトコルとして使用されます。この場合には、デフォルトのソケット・タイプのプロビジョンはありません。

SOCTYPE および PROTO の両方が非ゼロに指定されている場合、それらは SERVIC で指定された値にかかわらず、互換性がなければなりません。このコンテキストでは、互換とは以下のいずれかを意味します。

- SOCTYPE=SOCK_STREAM および PROTO=IPPROTO_TCP
- SOCTYPE=SOCK_DGRAM および PROTO=IPPROTO_UDP

- SOCTYPE が SOCK_RAW と指定されている場合、PROTO には任意の値が指定可能

PROTO

フルワード 2 進数フィールド。返される情報を特定のプロトコルに制限するために使用します。値 0 は、呼び出し元がどのプロトコルでも受け入れることを意味します。以下に示すのは、受け入れ可能なプロトコルです。

プロトコル名	**10** 進数値	説明
IPPROTO_TCP	6	TCP
IPPROTO_UDP	17	ユーザー・データグラム

SOCTYPE が 0 で、かつ PROTO が非ゼロである場合、PROTO の受け入れ可能な入力値は IPPROTO_TCP および IPPROTO_UDP のみです。それ以外の場合、GETADDRINFO 呼び出しは失敗し、戻りコード EAI_BADFLAGS が出されます。

PROTO および SOCTYPE の両方が 0 と指定されている場合、GETADDRINFO は以下のように進みます。

- SERVIC が NULL の場合、または SERVIC が数値の場合、返されるアドレス情報により、SOCTYPE の指定はデフォルトの SOCK_STREAM になります。
- SERVIC がサービス名として指定されている場合 (例えば、SERVIC=FTP)、GETADDRINFO は適切なサービス・ファイルを 2 回検索します。最初の検索では SOCK_STREAM がプロトコルとして使用され、2 番目の検索では SOCK_DGRAM がプロトコルとして使用されます。この場合には、デフォルトのソケット・タイプのプロビジョンはありません。

PROTO および SOCTYPE の両方が非ゼロに指定されている場合、それらは SERVIC で指定された値にかかわらず、互換性がなければなりません。このコンテキストでは、互換とは以下のいずれかを意味します。

- SOCTYPE=SOCK_STREAM および PROTO=IPPROTO_TCP
- SOCTYPE=SOCK_DGRAM および PROTO=IPPROTO_UDP
- SOCTYPE=SOCK_RAW、この場合には PROTO に任意の値が指定可能

SERVIC に指定された値の検索が失敗した場合 (例えば、サービス名が入力プロトコルを使用する適切なサービス・ファイルに存在していない場合)、GETADDRINFO 呼び出しは失敗し、戻りコード EAI_SERVICE が出されます。

NAMELEN

フルワード 2 進数フィールド。入力では、このフィールドは 0 であることが必要です。

CANNONNAME

フルワード 2 進数フィールド。入力では、このフィールドは 0 であることが必要です。

NAME
フルワード 2 進数フィールド。入力では、このフィールドは 0 であることが必要です。

NEXT フルワード 2 進数フィールド。入力では、このフィールドは 0 であることが必要です。

注:

- FLAGS は、その対応する 10 進数で指定できます。
- FLAGS のバイナリー OR を COBOL プログラムで実行する場合は、以下の例で示すとおり、必要な COBOL ステートメントを追加するだけで済みます。COBOL ADD の後の FLAGS フィールドの値が 10 進数 80 または X'00000050' であることに注意してください。これは AI_V4MAPPED および AI_ADDRCONFIG、または X'00000010' および X'00000040' での OR 実行の合計です。

```
01 AI-V4MAPPED PIC 9(8) BINARY VALUE 16.
01 AI-ADDRCONFIG PIC 9(8) BINARY VALUE 64.
ADD AI-V4MAPPED TO FLAGS.
ADD AI-ADDRCONFG TO FLAGS.
```

RES 最初はフルワード 2 進数フィールド。成功して戻ると、このフィールドには 1 つ以上のアドレス情報構造体のチェーンへのポインターが入ります。(PRD1.MACLIB からの) EZBREHST マクロを使用して、アドレス情報マッピングを設定します。この構造体は、呼び出し側アプリケーションのキーに割り振られます。複数のタスク間で、これらの構造体を使用したり参照したりしないでください。構造体の使用を終了した場合は、返されたポインターを TYPE=FREEADDRINFO 呼び出しで指定することで、そのストレージを明示的に解放します。明示的に解放されていないストレージは、タスクの終了時に解放されます。

CANNLEN
最初は入力パラメーター。RES CANONNAME フィールドにより返される正規名の長さを入れるために使用する、フルワード 2 進フィールド。これはオプションのフィールドです。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO フィールドは無視できます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ADDRINFO 構造体は、間接アドレッシングを使用して、可変の個数の NAMES を返します。この構造体は、PL/I またはアセンブラー言語を使用してコーディングす

る場合は、比較的簡単に処理できます。COBOL でコーディングする場合、この構造体を解釈するのは簡単ではありません。サブルーチン 330 ページの『EZACIC09』 を使用すると、GETADDRINFO 呼び出しによって戻される情報の解釈を単純化できます。

GETCLIENTID

GETCLIENTID 呼び出しは、呼び出し側プログラムの TCP/IP アドレス・スペースが呼び出し側アプリケーションを認識する識別子を戻します。

CLIENT パラメーターは、271 ページの『GIVESOCKET』 および 319 ページの『TAKESOCKET』 呼び出しで使用されます。

GETCLIENTID がサーバーによって呼び出されると、呼び出し元 (必ずしもクライアント とは限らない) の ID が返されます。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'GETCLIENTID'.
    01  CLIENT.
        03  DOMAIN      PIC 9(8) BINARY.
        03  NAME        PIC X(8).
        03  TASK        PIC X(8).
        03  RESERVED    PIC X(20).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION CLIENT ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、'GETCLIENTID' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

アプリケーションへ戻されるパラメーター値

CLIENT

呼び出しを発行したアプリケーションを記述する、クライアント ID 構造体です。

DOMAIN

フルワード 2 進数であり、クライアントのドメインを指定します。入力では、これは AF_INET のオプション・パラメーターであり、クライアントのドメインを指定するための AF_INET6 には必須パラメーターです。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2、または AF_INET6 を示す 10 進数の 19 に設定されます。出力では、これはクライアントの返されたドメインです。

NAME
8 バイトの文字フィールドです。パーティション ID から構成され、左寄せでブランクを入れます。

TASK
8 バイトの文字フィールドです。このタスク ID は、INITAPI 呼び出しによってユーザーが指定するか、システムでデフォルト設定されます (詳しくは、INITAPI 呼び出しの説明を参照してください)。

RESERVED
20 バイトの予約済み文字フィールドを指定します。このフィールドは必須であり、TCP/IP によって内部的に使用されます。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

GETHOSTBYADDR

GETHOSTBYADDR 呼び出しは、呼び出しに指定された IP アドレスを持つホストの、ドメイン・ネームおよび別名を戻します。

1 つの TCP/IP ホストは、複数の別名および複数の IP アドレスを持つことができます。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'GETHOSTBYADDR'.
    01  HOSTADDR        PIC 9(8) BINARY.
    01  HOSTENT         PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION HOSTADDR HOSTENT RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'GETHOSTBYADDR' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

HOSTADDR
フルワード 2 進数フィールドであり、名前を取得するホストの IP アドレ

ス (ネットワーク・バイト・オーダー) に設定されます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

アプリケーションへ戻されるパラメーター値

HOSTENT

フルワードのフィールドであり、HOSTENT 構造体のアドレスを含みます。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	エラーが発生しました。

GETHOSTBYADDR は、図 17 に示す HOSTENT 構造体を戻します。

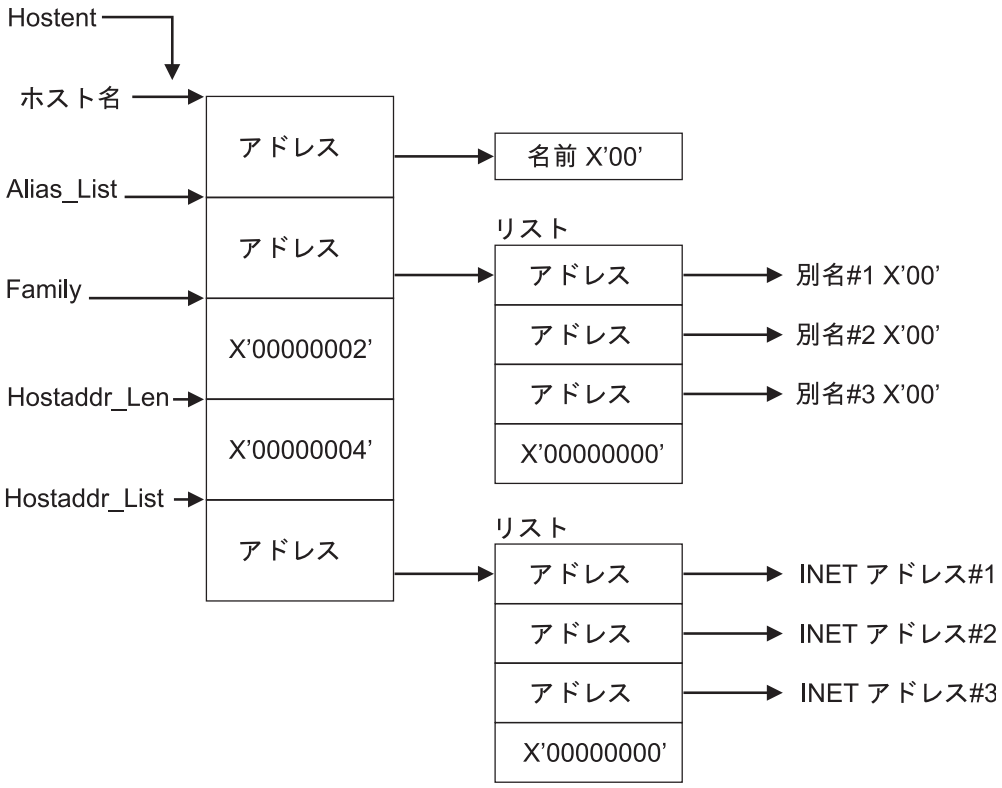


図 *17. GETHOSTBYADDR* 呼び出しによって戻される *HOSTENT* 構造体

この構造体に含まれるのは次のとおりです。

- 呼び出しによって戻されるホスト名のアドレス。名前は可変長であり、X'00' で終わります。
- 呼び出しによって戻される別名を指すアドレス・リストのアドレス。このリストは、ポインター X'00000000' で終わります。それぞれの別名は可変長フィールドであり、X'00' で終わります。

重要: TCP/IP for z/VSE では別名はサポートされません。

- FAMILY フィールドに戻される値は、常に AF_INET に対応する 2 です。
- HOSTADDR_LEN フィールドに戻されるホスト IP アドレスの長さは、常に AF_INET に対応する 4 です。
- 呼び出しによって戻されるホスト IP アドレスを指すアドレス・リストのアドレス。リストは、ポインター X'00000000' で終わります。

HOSTENT 構造体は、間接アドレッシングを使用して、可変の個数の別名および IP アドレスを戻します。この構造体は、PL/I またはアセンブラー言語を使用してコーディングする場合は、比較的簡単に処理できます。COBOL でコーディングする場合、この構造体を解釈するのは簡単ではありません。サブルーチン EZACIC08 を利用すると、GETHOSTBYADDR および GETHOSTBYNAME 呼び出しによって戻される情報の解釈を単純化できます。EZACIC08 について詳しくは、327 ページの『EZACIC08』を参照してください。

GETHOSTBYNAME

GETHOSTBYNAME 呼び出しは、呼び出しに指定されたドメイン・ネームを持つホストの、別名および IP アドレスを戻します。

1 つの TCP/IP ホストは、複数の別名および複数の IP アドレスを持つことができます。

TCP/IP は、ネーム・サーバーが存在すれば、ネーム・サーバーを介してホスト名を解決しようとします。シンボル名を IP アドレスに変換するために呼び出しが行われると、TCP/IP for z/VSE は、ローカル名テーブル (DEFINE NAME によって作成される) を最初に検索します。この検索が失敗した場合、名前は指定された DNS (SET DNSx で設定される) に渡されます。TCP/IP for z/VSE は、応答を受け取るか、すべてのサーバーにポーリングをし終わるまで、DNS1 から始めて各 DNS を試します。応答する最初のサーバーが、要求が成功か失敗かを決定します。ある DNS 内での検索が失敗した場合、デフォルトのドメイン・ストリング (SET DEFAULT_DOMAIN で指定される) が名前に (ピリオドの後に続けて) 付加され、ネーム解決のために最後にその DNS に情報が求められます。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'GETHOSTBYNAME'.
    01  NAMELEN         PIC 9(8)  BINARY.
    01  NAME            PIC X(24).
    01  HOSTENT         PIC 9(8)  BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION NAMELEN NAME
                     HOSTENT RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'GETHOSTBYNAME' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

NAMELEN
ホスト名の長さを示す値に設定されます。

NAME
24 文字までの文字ストリングで、ホスト名に設定されます。この呼び出しは、この名前に対する HOSTENT 構造体のアドレスを戻します。

アプリケーションへ戻されるパラメーター値

HOSTENT
フルワード 2 進数フィールドであり、HOSTENT 構造体のアドレスを含みます。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	エラーが発生しました。

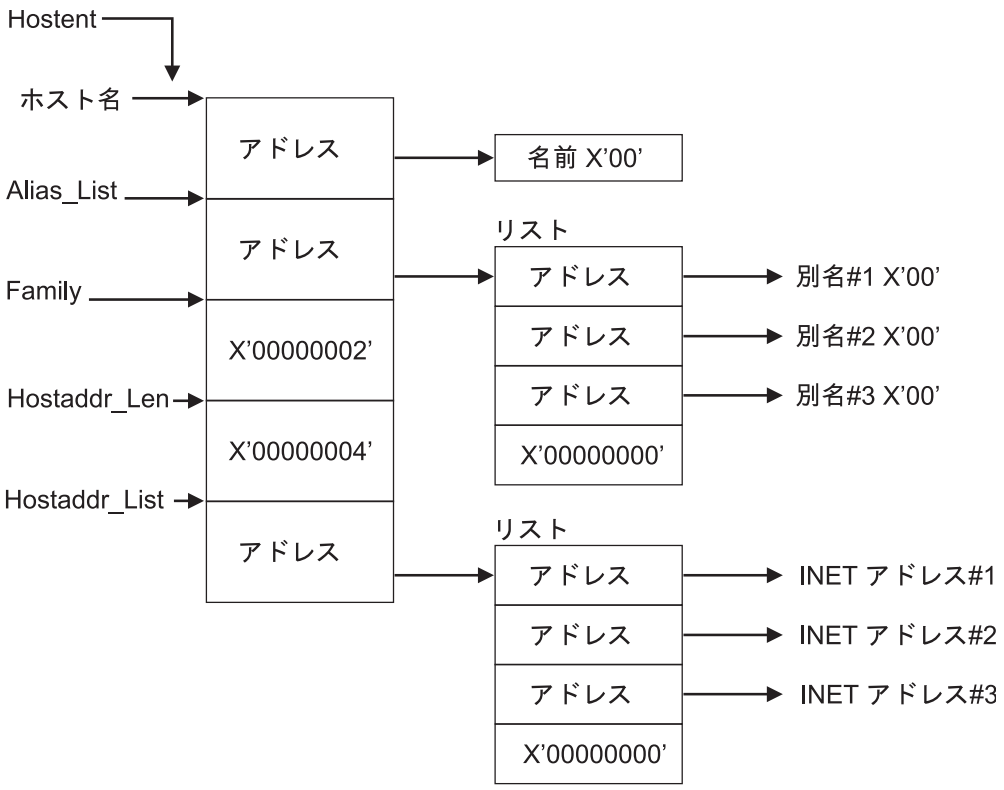


図 *18. GETHOSTYBYNAME* 呼び出しによって戻される *HOSTENT* 構造体

GETHOSTBYNAME は、 256 ページの図 18 に示す HOSTENT 構造体を戻します。この構造体に含まれるのは次のとおりです。

- 呼び出しによって戻されるホスト名のアドレス。名前は可変長であり、X'00' で終わります。
- 呼び出しによって戻される別名を指すアドレス・リストのアドレス。このリストは、ポインター X'00000000' で終わります。それぞれの別名は可変長フィールドであり、X'00' で終わります。

  重要**:** TCP/IP for z/VSE では別名はサポートされません。
- FAMILY フィールドに戻される値は、常に AF_INET に対応する 2 です。
- HOSTADDR_LEN フィールドに戻されるホスト IP アドレスの長さは、常に AF_INET に対応する 4 です。
- 呼び出しによって戻されるホスト IP アドレスを指すアドレス・リストのアドレス。リストは、ポインター X'00000000' で終わります。

HOSTENT 構造体は、間接アドレッシングを使用して、可変の個数の別名および IP アドレスを戻します。この構造体は、PL/I またはアセンブラー言語を使用してコーディングする場合は、比較的簡単に処理できます。COBOL でコーディングする場合、この構造体を解釈するのは簡単ではありません。サブルーチン EZACIC08 を利用すると、GETHOSTBYADDR および GETHOSTBYNAME 呼び出しによって戻される情報の解釈を単純化できます。EZACIC08 について詳しくは、 327 ページの『EZACIC08』を参照してください。

GETHOSTID

GETHOSTID 呼び出しは、現行ホストの 32 ビットの IP アドレスを戻します。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'GETHOSTID'.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、'GETHOSTID' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

RETCODE

ホストの 32 ビットの IP アドレスが入ったフルワード 2 進数フィールドを戻します。RETCODE が ***-1*** である場合、エラーがあったことを示します。考えられる理由は、TCP/IP が開始されていないことです。この呼び出しには ERRNO パラメーターはありません。

GETHOSTNAME

GETHOSTNAME 呼び出しは、ローカル・ホストのドメイン・ネームを戻します。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'GETHOSTNAME'.
    01  NAMELEN         PIC 9(8) BINARY.
    01  NAME            PIC X(24).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION NAMELEN NAME
                     ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、GETHOSTNAME を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

NAMELEN

フルワード 2 進数フィールドであり、NAME フィールドの長さに設定されます。

アプリケーションへ戻されるパラメーター値

NAMELEN

フルワード 2 進数フィールドであり、ホスト名の長さに設定されます。

NAME

ホスト名を受け取るフィールドを指示します。TCP/IP for z/VSE で許容される最大長は 64 文字です。インターネット標準の最大長は 255 文字です。NAME フィールドの実際の長さは、NAMELEN に入っています。

ERRNO

フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

GETIBMOPT

GETIBMOPT 呼び出しは、特定の z/VSE システムにインストールされている TCP/IP イメージの数、その状況、バージョン、および名前を返します。

この情報に基づき、呼び出し側は INITAPI 呼び出しを使用して、接続先とする TCP/IP イメージを動的に選択できます。

重要: この関数呼び出しは TCP/IP for z/VSE では使用できません。

COBOL の例

```
    WORKING-STORAGE SECTION.
   01 SOC-FUNCTION PIC X(16) VALUE IS 'GETIBMOPT'.
   01 COMMAND PIC 9(8) BINARY VALUE IS 1.
   01 BUF.
    03 NUM-IMAGES PIC 9(8) COMP.
    03 TCP-IMAGE OCCURS 8 TIMES.
     05 TCP-IMAGE-STATUS PIC 9(4) BINARY.
     05 TCP-IMAGE-VERSION PIC 9(4) BINARY.
     05 TCP-IMAGE-NAME PIC X(8)
   01 ERRNO PIC 9(8) BINARY.
   01 RETCODE PIC S9(8) BINARY.
PROCEDURE DIVISION.
   CALL 'EZASOKET' USING SOC-FUNCTION COMMAND BUF ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、GETIBMOPT を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

COMMAND
処理するコマンドを指定する値またはフルワード 2 進数のアドレス。有効な値は 1 のみです。

アプリケーションへ戻されるパラメーター値

BUF アクティブな各 TCP/IP イメージの状況、バージョン、および名前の格納先の 100 バイト・バッファーです。

正常終了して返された場合、このバッファー・エントリーには最大 8 つのアクティブな TCP/IP イメージの状況、名前、およびバージョンが入っています。以下のレイアウトは、呼び出しが完了したときの BUF フィールドを示しています。NUM_IMAGES フィールドは、合計 BUF フィールドに含まれている TCP_IMAGE の得エントリー数を示します。返された NUM_IMAGES が 0 の場合、TCP/IP イメージは存在しません。

状況フィールド
意味

X'8xxx'
アクティブ

X'4xxx'
終了中

X'2xxx'
ダウン

X'1xxx'
停止または停止中

注: 上記の状況フィールドの xxx は IBM 用に予約されており、これに任意の値を含めることができます。

GETNAMEINFO

GETNAMEINFO 呼び出しは、マクロで指定されたソケット・アドレスのノード名およびサービス・ロケーションを返します。

正常に完了した場合、GETNAMEINFO は、要求されていれば、指定したバッファーで命名されたノードおよびサービスを返します。

重要: この関数呼び出しは TCP/IP for z/VSE では使用できません。

COBOL の例

```
WORKING-STORAGE SECTION.
    01 SOC-FUNCTION PIC X(16) VALUE IS 'GETNAMEINFO'.
    01 NAMELEN PIC 9(8) BINARY.
    01 HOST PIC X(255).
    01 HOSTLEN PIC 9(8) BINARY.
    01 SERVIC PIC X(32).
    01 SERVLEN PIC 9(8) BINARY.
    01 FLAGS PIC 9(8) BINARY VALUE 0.
    01 NI-NOFQDN PIC 9(8) BINARY VALUE 1.
    01 NI-NUMERICHOST PIC 9(8) BINARY VALUE 2.
    01 NI-NAMEREQD PIC 9(8) BINARY VALUE 4.
    01 NI-NUMERICSERVER PIC 9(8) BINARY VALUE 8.
    01 NI-DGRAM PIC 9(8) BINARY VALUE 16.
    01 NI-NUMERICSCOPE PIC 9(8) BINARY VALUE 32.

* IPv4 socket structure.
    01 NAME.
       03 FAMILY PIC 9(4) BINARY.
       03 PORT PIC 9(4) BINARY.
       03 IP-ADDRESS PIC 9(8) BINARY.
       03 RESERVED PIC X(8).

* IPv6 socket structure.
    01 NAME.
       03 FAMILY PIC 9(4) BINARY.
       03 PORT PIC 9(4) BINARY.
       03 FLOWINFO PIC 9(8) BINARY.
       03 IP-ADDRESS.
          10 FILLER PIC 9(16) BINARY.
          10 FILLER PIC 9(16) BINARY.
       03 SCOPE-ID PIC 9(8) BINARY.
    01 ERRNO PIC 9(8) BINARY.
    01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.
    MOVE 28 TO NAMELEN.
    MOVE 255 TO HOSTLEN.
    MOVE 32 TO SERVLEN.
    MOVE NI-NAMEREQD TO FLAGS.
    CALL 'EZASOKET' USING SOC-FUNCTION NAME NAMELEN HOST
         HOSTLEN SERVIC SERVLEN FLAGS ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、GETNAMEINFO を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

NAME

変換される IPv4 または IPv6 ソケット・アドレス構造体。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。

AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。IPv4 ソケット・アドレス構造体は、以下のフィールドを指定する必要があります。

フィールド
: 説明

FAMILY
: ハーフワード 2 進数であり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
: ハーフワード 2 進数であり、ポート番号を指定します。

IPv4-ADDRESS
: フルワード 2 進数であり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
: 8 バイトの予約フィールドです。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを指定する必要があります。

フィールド
: 説明

NAMELEN
: IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
: IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
: ハーフワード 2 進数であり、ポート番号を指定します。

FLOW-INFO
このフィールドは TYPE=GETNAMEINFO マクロにより無視されます。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、128 ビットの IPv6 インターネット・アドレスをネットワーク・バイト・オーダーで指定します。

SCOPE-ID
フルワード 2 進数フィールドであり、インターフェース・インデックスとしての IPv6 アドレスの有効範囲を指定します。
IPv6-ADDRESS のアドレスがリンク・ローカル・アドレスであり、さらに HOST パラメーターも指定されているのでない限り、リゾルバーは SCOPE_ID フィールドを無視します。

NAMELEN
フルワード 2 進数フィールド。NAME 引数により示されるソケット・アドレス構造体の長さ。

HOST
入力では、返された解決済みホスト名を保持できるストレージであり、入力ソケット・アドレスに対して最大で 255 バイトの長さにすることができます。解決済みホスト名を入れるために不適切なストレージが指定された場合、リゾルバーは指定されたストレージまでのホスト名を返し、切り捨てが行われる場合もあります。ホスト名が見つからない場合、その名前の代わりに数値形式のホスト・アドレスが返されます。ただし、NI_NAMEREQD オプションが指定されているが、ホスト名が見つからない場合は、エラーが返されます。以下のパラメーターのグループのうち、1 つか両方が必要です。

- HOST および HOSTLEN パラメーター
- SERVIC および SERVLEN パラメーター

そうでない場合には、エラーが発生します。照会される HOST 名は、HOSTLEN まで、または最初の 2 進 0 までで構成されます。

IPv6-ADDRESS 値がリンク・ローカル・アドレスであり、SCOPE_ID インターフェース・インデックスが非ゼロである場合、スコープ情報が、*host%scope information* 形式を使用して解決済みホスト名に追加されます。スコープ情報は、数値形式の SCOPE_ID インターフェース・インデックス、または SCOPE_ID インターフェース・インデックスに関連付けられたインターフェース名にすることができます。返される形式を選択するには、NI_NUMERICSCOPE オプションを使用します。結合されたホスト名とスコープ情報は、255 バイト以下です。

HOSTLEN
フルワード 2 進数フィールドであり、返された解決済みホスト名を入れるために使用されるホスト・ストレージの長さが入ります。入力で HOSTLEN が 0 の場合、解決済みホスト名は返されません。HOSTLEN 値は、返される最長のホスト名、またはホスト名とスコープ情報の組み合わせの長さ以上でなければなりません。TYPE=GETNAMEINFO は、ホスト名、またはホスト名とスコープ情報の組み合わせを、HOSTLEN 値により指定された長さまで返します。出力では、HOSTLEN には、返されるホス

ト名、またはホスト名とスコープ情報の組み合わせの長さが入ります。これはオプション・フィールドですが、このフィールドを指定する場合は、HOST 値をコーディングする必要もあります。以下のパラメーターのグループのうち、1 つか両方が必要です。

- HOST および HOSTLEN パラメーター
- SERVIC および SERVLEN パラメーター

そうでない場合には、エラーが発生します。

SERVIC

入力では、返された解決済みサービス名を保持できるストレージであり、入力ソケット・アドレスに対して最大で 32 バイトの長さにすることができます。解決済みサービス名を入れるために不適切なストレージが指定された場合、リゾルバーは指定されたストレージまでサービス名を返します。さらに、切り捨てが行われる場合もあります。サービス名が見つからない場合、または NI_NUMERICSERV が FLAGS オペランドに指定されている場合、その名前の代わりに表示形式のサービス・アドレスが返されます。これはオプション・フィールドですが、このフィールドを指定する場合は、SERVLEN パラメーターをコーディングする必要もあります。照会される SERVIC 名は、SERVLEN まで、または最初の 2 進ゼロまでで構成されます。以下のパラメーターのグループのうち、1 つか両方が必要です。

- HOST および HOSTLEN パラメーター
- SERVIC および SERVLEN パラメーター

そうでない場合には、エラーが発生します。

SERVLEN

最初は入力パラメーター。フルワード 2 進数フィールドであり、返された解決済みサービス名を入れるために使用される SERVIC ストレージの長さが入ります。入力で SERVLEN が 0 の場合、サービス名情報は返されません。SERVLEN は、返される最長のサービス名の長さ以上でなければなりません。TYPE=GETNAMEINFO は、SERVLEN により指定された長さまでサービス名を返します。出力では、SERVLEN には、返される解決済みサービス名の長さが入ります。これはオプション・フィールドですが、これを指定する場合は、SERVIC パラメーターもコーディングする必要があります。以下のパラメーターのグループのうち、1 つか両方が必要です。

- HOST および HOSTLEN パラメーター
- SERVIC および SERVLEN パラメーター

そうでない場合には、エラーが発生します。

FLAGS

フルワード 2 進数フィールド。これはオプションのフィールドです。FLAGS 引数は、以下のリテラル値またはフルワード 2 進数フィールド値にすることができます。

リテラル値	バイナリー値	**10** 進数値	説明
'NI_NOFQDN'	X'00000001'	1	完全修飾ドメイン・ネームの NAME 部分を返します。

リテラル値	バイナリー値	**10** 進数値	説明
'NI_NUMERICHOST'	X'00000002'	2	数値形式のホスト・アドレスのみを返します。
'NI_NAMEREQD'	X'00000004'	4	ホスト名が見つからない場合にエラーを返します。
'NI_NUMERICSERV'	X'00000008'	8	数値形式のサービス・アドレスのみを返します。
'NI_DGRAM'	X'00000010'	16	サービスがデータグラム・サービスであることを示します。デフォルトの動作では、サービスをストリーム・サービスと想定します。
'NI_NUMERICSCOPE'	X'00000020'	32	該当する場合、数値形式の SCOPE-ID インターフェース・インデックスのみを返します。

アプリケーションへ戻されるパラメーター値

ERRNO

フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

GETPEERNAME

GETPEERNAME 呼び出しは、ローカル・ソケットが接続されている先のリモート・ソケットの名前を戻します。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'GETPEERNAME'.
    01  S               PIC 9(4) BINARY.
    01  NAME.
        03  FAMILY      PIC 9(4) BINARY.
        03  PORT        PIC 9(4) BINARY.
        03  IP-ADDRESS  PIC 9(8) BINARY.
        03  RESERVED    PIC X(8).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、GETPEERNAME を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S
ハーフワード 2 進数であり、アドレスを取得したいリモート・ピアに接続されたローカル・ソケットのソケット記述子に設定されます。

アプリケーションへ戻されるパラメーター値

NAME
最初は、ピア名構造体を指します。これには呼び出し完了時に、S で示されたローカル・ソケットに接続されたリモート・ソケットの IPv4 または IPv6 アドレス構造体が入ります。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
説明

FAMILY
ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、クライアントのポート番号を指定します。

IPv4-ADDRESS
フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
説明

NAMELEN
IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、クライアントのポート番号に設定されます。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

GETSOCKNAME

GETSOCKNAME 呼び出しは、指定されたソケットに現在バインドされているアドレスを戻します。

ソケットが現在アドレスにバインドされていない場合、この呼び出しは、FAMILY フィールドが設定され、構造セットの残りが 0 に設定された状態で戻ります。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

ストリーム・ソケットには、BIND、CONNECT、または ACCEPT のいずれかの呼び出しが成功するまでは名前が割り当てられません。したがって、暗黙のバインドの後に GETSOCKNAME 呼び出しを使用して、ソケットに割り当てられたポートを検出できます。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'GETSOCKNAME'.
    01  S               PIC 9(4) BINARY.
    01  NAME.
        03  FAMILY      PIC 9(4) BINARY.
        03  PORT        PIC 9(4) BINARY.
        03  IP-ADDRESS  PIC 9(8) BINARY.
        03  RESERVED    PIC X(8).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、GETSOCKNAME を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S

ハーフワード 2 進数であり、アドレスを取得したいローカル・ソケットの記述子に設定されます。

アプリケーションへ戻されるパラメーター値

NAME

最初は、アプリケーションが IPv4 または IPv6 ソケット・アドレス構造体へのポインターを設定し、呼び出しの完了時に、その構造体にソケット名が入ります。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
: 説明

FAMILY

ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、このソケットにバインドされたポート番号を指定します。ソケットがバインドされていない場合、ゼロが戻されます。

IPv4-ADDRESS
フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
説明

NAMELEN
IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、このソケットにバインドされたポート番号を指定します。ソケットがバインドされていない場合、ゼロが戻されます。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

GETSOCKOPT

GETSOCKOPT 呼び出しは、 311 ページの『SETSOCKOPT』 呼び出しによって設定されたオプションを照会します。各ソケットには、いくつかのオプションが関連付けられています。それらのオプションの説明は下にあります。GETSOCKOPT 呼び出しを発行するときには、照会するオプションを指定する必要があります。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'GETSOCKOPT'.
    01  S               PIC 9(4) BINARY.
    01  OPTNAME         PIC 9(8) BINARY.
        88  SO-LINGER     VALUE  128.
    01  OPTVAL          PIC X(16) BINARY.
    01  OPTLEN          PIC 9(8) BINARY.
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S OPTNAME
                    OPTVAL OPTLEN ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、GETSOCKOPT を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S

ハーフワード 2 進数であり、オプションを照会するソケットのソケット記述子を指定します。

OPTNAME

GETSOCKOPT を発行する前に、必須指定のオプションを OPTNAME に設定してください。オプションは次のとおりです。

IP_MULTICAST_IF

このオプションは、ソケット・アプリケーションからアウトバウンド・マルチキャスト・データグラムを送信する際に使用する IPv4 インターフェース・アドレスを取得するために使用します。これは IPv4 専用のソケット・オプションです。

注**:** マルチキャスト・データグラムを送信できるインターフェースは、一度に 1 つのみです。

IP_MULTICAST_LOOP

送信ホスト自体が属するグループに送信されたマルチキャスト・デ

ータグラムについて、マルチキャスト・データグラムのコピーをループバックするかどうかを指定するには、このオプションを使用します。デフォルトでは、データグラムをループバックします。これは IPv4 専用のソケット・オプションです。

IP_MULTICAST_TTL
このオプションを使用して、発信マルチキャスト・データグラムの IP データグラム存続時間 (ホップ) を取得します。デフォルト値は '01'x です。これは、マルチキャストがローカル・サブネットでのみ使用可能であることを意味します。これは IPv4 専用のソケット・オプションです。

IPV6_MULTICAST_HOPS
このオプションを使用して、発信マルチキャスト・パケットに使用するホップ限界を取得します。これは IPv6 専用のソケット・オプションです。

IPV6_MULTICAST_IF
このオプションを使用して、ソケット・アプリケーションからアウトバウンド・マルチキャスト・データグラムを送信する際に使用する IPv6 インターフェースの索引を取得します。これは IPv6 専用のソケット・オプションです。

IPV6_MULTICAST_LOOP
送信ホスト自体が属するグループに対してデータグラムを送信する場合、ローカル配信のため、IP レイヤーによって発信インターフェース上でマルチキャスト・データグラムをループバックするかどうかを指定するには、このオプションを使用します。デフォルトでは、マルチキャスト・データグラムをループバックします。これは IPv6 専用のソケット・オプションです。

IPV6_UNICAST_HOPS
このオプションを使用して、発信ユニキャスト IPv6 パケットに使用するホップ限界を取得します。これは IPv6 専用のソケット・オプションです。

IPV6_V6ONLY
このオプションを使用して、ソケットを IPv6 パケットのみの送受信に制限するかどうかを指定します。デフォルトでは、IPv6 パケットのみの送受信に制限しません。これは IPv6 専用のソケット・オプションです。

SO-LINGER
LINGER の状況を要求します。

- LINGER オプションが使用可能にされていて、データ伝送が完了していない場合、CLOSE 呼び出しは、データが送信されるか、接続がタイムアウトになるまで、呼び出し側プログラムをブロックします。
- LINGER が使用可能にされていない場合、CLOSE 呼び出しは、呼び出し側プログラムをブロックせずに戻ります。TCP/IP はデータを送信しようとします。データ転送は通常は成功しますが、

TCP/IP は指定された時間だけしかデータを送信しようとしないため、成功が保証されているわけではありません。

アプリケーションへ戻されるパラメーター値

OPTVAL

- OPTNAME に SO-LINGER が指定されている場合、次の構造体が戻されます。

```
ONOFF       PIC X(8)
LINGER      PIC 9(8)
```

  - ONOFF にゼロ以外の値が戻された場合、このオプションが使用可能になっていることを示します。ゼロ値は、使用不可になっていることを示します。
  - LINGER 値は、CLOSE が発行された後に TCP/IP がデータ送信を続行する時間 (秒) を示します。Linger 時間の設定 方法については、 311 ページの『SETSOCKOPT』を参照してください。

OPTLEN

フルワード 2 進数フィールドであり、OPTVAL に戻されるデータの長さを含みます。

- OPTNAME が SO-LINGER の場合、OPTVAL は 2 個のフルワードを含むので、OPTLEN は 8 (2 個のフルワード) に設定されます。

ERRNO

フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

GIVESOCKET

GIVESOCKET 呼び出しは、あるプロセスから別のプロセスにソケットを渡すのに使用されます。

通常、TCP/IP は、以下のシーケンスで GETCLIENTID、GIVESOCKET、および TAKESOCKET 呼び出しを使用します。

1. あるプロセスが GETCLIENTID 呼び出しを発行して、そのプロセスの領域のジョブ名、および VSE サブタスク ID を取得します。この情報は、GIVESOCKET 呼び出しに使用されます。
2. そのプロセスは GIVESOCKET 呼び出しを発行して、子プロセスが使用するソケットを準備します。
3. 子プロセスは TAKESOCKET 呼び出しを発行して、準備されたソケットを取得します。これで、ソケットは子プロセスに所属し、TCP/IP によって別プロセスとの通信に使用できるようになります。

注: TAKESOCKET 呼び出しは、新しいソケット記述子を RETCODE に戻します。子プロセスは、この新しいソケット記述子を、このソケットを使用するすべての呼び出しに使用しなければなりません。TAKESOCKET 呼び出しに渡されたソケット記述子を使用してはなりません。

4. GIVESOCKET コマンドの発行後、親プロセスは SELECT コマンドを発行します。このコマンドは子がソケットを取得するのを待ちます。
5. 子がソケットを取得すると、親は例外条件を受け取って、SELECT コマンドが解放されます。
6. 親プロセスはソケットをクローズします。

これ以降、親は元のソケット記述子を再使用できます。

重要: IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'GIVESOCKET'.
    01  S               PIC 9(4) BINARY.
    01  CLIENT.
        03  DOMAIN      PIC 9(8) BINARY.
        03  NAME        PIC X(8).
        03  TASK        PIC X(8).
        03  RESERVED    PIC X(20).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S CLIENT ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'GIVESOCKET' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S
ハーフワード 2 進数であり、与えられるソケットのソケット記述子に設定されます。

CLIENT
ソケットが与えられる先のアプリケーションの ID を含む構造体です。

DOMAIN
フルワード 2 進数であり、クライアントのドメインを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2、または AF_INET6 を示す 10 進数の 19 に設定されます。

注: GIVESOCKET で指定されたソケットは、同じ DOMAIN、アドレス・ファミリー (AF_INET または AF_INET6) の TAKESOCKET によってのみ取得できます。

NAME

8 文字のフィールドであり、左寄せで右にはブランクを入れます。ソケットを取得するアプリケーションのアドレス・スペース名 (パーティション ID) に設定されます。このフィールドがブランクのままである場合、任意の z/VSE パーティションがソケットを取得できます。

TASK

8 文字のフィールドを指定し、ブランクに設定するか、またはソケットを取得する VSE サブタスクの ID に設定できます。このフィールドがブランクに設定されている場合、NAME フィールドに指定されたパーティション内の任意のサブタスクがソケットを取得できます。

RESERVED

20 バイトの予約フィールドです。このフィールドは必須ですが、内部的にのみ使用されます。

アプリケーションへ戻されるパラメーター値

ERRNO

フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

GSKFREEMEM

GSKFREEMEM 呼び出しは、前の SSL 関数呼び出しでアプリケーションに渡されたメモリーを解放します。

注: GSKGETDNBYLAB 呼び出しによってヌル終了ストリングで戻された識別名は、GSKFREEMEM を使用して解放される必要があります。

COBOL の例

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION  PIC X(16) VALUE 'GSKFREEMEM      '.
    01  AREA          PIC 9(8) BINARY.
    01  ERRNO         PIC 9(8) BINARY.
    01  RETCODE       PIC S9(8) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION AREA
                ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'GSKFREEMEM' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

AREA

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。詳しいエラー情報を示します。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

0 呼び出しは成功しました。

0 より小
エラーが発生しました。

GSKGETCIPHINF

GSKGETCIPHINF 呼び出しは、SSL for VSE の暗号関連情報を要求します。

この情報は、システムがサポートできる暗号化レベルを判別し、SSL が使用できる暗号仕様のリストを戻します。これにより、アプリケーションはその実行時に、インストール済みアプリケーションが要求できる SSL 暗号化のレベルを判別することができます。

COBOL の例

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION  PIC X(16) VALUE 'GSKGETCIPHINF   '.
    01  CIPHLEVEL     PIC 9(8) BINARY.
    01  SECLEVEL      PIC X(104).
    01  ERRNO         PIC 9(8) BINARY.
    01  RETCODE       PIC S9(8) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION CIPHLEVEL SECLEVEL
                ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'GSKGETCIPHINF' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

CIPHLEVEL
フルワード 2 進数フィールドであり、戻される暗号情報のタイプを決定する数値に設定されます。有効値は、次のものです。

1 米国/カナダ外で使用可能な暗号情報のみが戻されます (GSK_LOW_SECURITY)。

2 米国/カナダ外で使用可能な暗号情報と米国/カナダ国内用の暗号情報が戻されます (GSK_HIGH_SECURITY)。

アプリケーションへ戻されるパラメーター値

SECLEVEL

104 バイトの領域であり、ここにシステムが以下の情報を戻します。

4 バイト
システム SSL バージョン (常に GSK_VERSION3 を示す 3)

64 バイト
文字ストリング (x00 で終わる) であり、システムで使用が許可される SSL バージョン 3 暗号仕様を含みます (これらの仕様は GSKSSOCINIT 呼び出しの V3CIPHER パラメーターに渡すことができます)。

32 バイト
SSL for VSE は SSL バージョン 2 暗号仕様をサポートしないため、このフィールドには常に 2 進ゼロが埋められます。

4 バイト
以下のいずれかです。

1 GSK_SEC_LEVEL_US

2 GSK_SEC_LEVEL_EXPORT

3 GSK_SEC_LEVEL_EXPORT_FR

ERRNO

フルワード 2 進数フィールド。詳しいエラー情報を示します。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

0 呼び出しは成功しました。

0 より小
エラーが発生しました。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

GSKGETDNBYLAB

GSKGETDNBYLAB 呼び出しは、ある鍵の完全な識別名を、キー・データベース・ファイル内のその鍵のラベルに基づいて戻します。

この値は、GSKSSOCINIT 呼び出しの DNAME フィールドに使用できます。

注: ヌル終了ストリングで戻される識別名は、GSKFREEMEM 呼び出しを使用して解放される必要があります。

COBOL の例

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION  PIC X(16) VALUE 'GSKGETDNBYLAB   '.
    01  KEYLABEL      PIC X(Length of key label).
    01  ERRNO         PIC 9(8) BINARY.
    01  RETCODE       PIC S9(8) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION KEYLABEL
                ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'GSKGETDNBYLAB' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

KEYLABEL

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。詳しいエラー情報を示します。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

0 より大
呼び出しは成功しました。RETCODE は、識別名が入った文字ストリングへのポインターを表します。

0 以下
呼び出しは失敗しました。

GSKINIT

GSKINIT 呼び出しは、現行パーティションについて、全般的な SSL for VSE 環境を設定します。

この関数が正常に完了した後、アプリケーションは、SSL for VSE インターフェースの呼び出し、およびセキュア・ソケット接続の作成と使用を行うことができます。

COBOL の例

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION  PIC X(16) VALUE 'GSKINIT         '.
    01  SECTYPE.
        05 SECTYPE1   PIC X(5) VALUE IS 'SSL30'.
        05 SECTPYE2   PIC 9(1) BINARY VALUE 0.
    01  KEYRING.
        05 KEYRING1   PIC X(11) VALUE IS 'PRIMARY.GSK'.
        05 KEYRING2   PIC 9(1) COMP VALUE 0.
    01  V3TIMEOUT     PIC 9(8) COMP VALUE 86400.
    01  CAROOTS       PIC 9(8) COMP VALUE 0.
    01  AUTHTYPE      PIC 9(8) COMP VALUE 0.
    01  ERRNO         PIC 9(8) BINARY.
```

```
    01  RETCODE       PIC S9(8) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION SECTYPE KEYRING
                V3TIMEOUT CAROOTS AUTHTYPE
                ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、'GSKINIT' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

SECTYPE

受け入れ可能な最小限のセキュリティー・プロトコルを示す文字ストリングです。値は大文字で入力し、X00 で終わらせなければなりません。有効な値は次のとおりです (二重引用符は除きます)。

- SSL バージョン 3.0 を表す "SSL30"
- TLS バージョン 1.0 を表す "TLS31" (クライアント・アプリケーションにはサポートされません)

KEYRING

秘密鍵および証明書が保管されている "lib.sublib" を指定する文字ストリングです。ストリングは x00 で終わらなければなりません。8 個のブランクからなるストリングにすると、プロシージャー $SSL4VSE.PROC に定義されたデフォルトの "SSL for VSE" ファイルを使用できます。このプロシージャーについて詳しくは、「*TCP/IP for VSE Optional Features*」を参照してください。

V3TIMEOUT

SSL V3 セッション ID の有効期限を表す秒数です。有効範囲は 0 から 86400 (1 日) です。このパラメーターが指定されていない場合、デフォルト値の 86400 が適用されます。

CAROOTS

証明書の検証に使用する CA (認証局) ルートを指定する値です。以下の値がサポートされます。

0 ローカル・キー・データベース・ファイルからの CA ルートを使用して、証明書を検証します。

1 VSE と同じ認証局によって発行された証明書でのクライアント認証を許可します。

AUTHTYPE

クライアントの証明書を検証するのに使用する方式を指定する値です。このフィールドは、CAROOTS が 1 に設定されている場合のみ使用されます。以下の値がサポートされます。

0 クライアントの証明書は、ローカル・キー・データベース・ファイルを使用して検証されます。

1 値 0 と同じ意味です。

2 値 0 と同じ意味です。

3 クライアントの証明書は検証されません。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。詳しいエラー情報を示します。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

0 呼び出しは成功しました。

0 以外
エラーが発生しました。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

GSKSSOCCLOSE

GSKSSOCCLOSE 呼び出しは、セキュア・ソケット接続を終了し、その接続用のすべての SSL for VSE リソースを解放します。

COBOL の例

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION  PIC X(16) VALUE 'GSKSSOCCLOSE    '.
    01  SSOCDATA      PIC 9(8) BINARY.
    01  ERRNO         PIC 9(8) BINARY.
    01  RETCODE       PIC S9(8) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION SSOCDATA
                ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'GSKSSOCCLOSE' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

SSOCDATA
GSKSSOCINIT 呼び出しによって RETCODE に戻される、GSKSOCDATA 構造体のアドレスです。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。詳しいエラー情報を示します。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

0 呼び出しは成功しました。

0 より小
エラーが発生しました。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

GSKSSOCINIT

GSKSSOCINIT 呼び出しは、SSL for VSE がセキュア・ソケット接続を開始または受け入れるために必要なデータ域を初期設定します。

この関数が正常に完了すると、セキュア・ソケット制御ブロック (これ以降、GSKSOCDATA で表します) へのポインターがアプリケーションに戻されます。このセキュア・ソケット接続を使用する他の呼び出しは、このポインターを参照する必要があります。

呼び出し中に、GSKSSOCINIT 呼び出しに指定された入力に基づいて、完全なハンドシェークが実行されます。SSL ハンドシェークの手順は SSL for VSE によって実行されますが、SSL ハンドシェーク中の SSL データの転送には、それに続くすべての読み取り/書き込み操作と同様、「通常の」RECV ルーチンと SEND ルーチン (EZAAPI 処理環境によって提供される) が使用されます。

COBOL の例

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION  PIC X(16) VALUE 'GSKSSOCINIT     '.
    01  S             PIC 9(4) BINARY.
    01  HANDSHAKE     PIC 9(8) BINARY.
    01  DNAME.
        05 DNAME1     PIC X(n) VALUE IS '.......'.
        05 DNAME2     PIC 9(1) BINARY VALUE 0.
    01  V3CIPHER.
        05 V3CIPHER1  PIC X(6) VALUE IS '0A0908'.
        05 V3CIPHER2  PIC 9(1) COMP VALUE 0.
    01  SECTYPE       USAGE IS POINTER.
    01  V3CIPHSEL     PIC X(2).
    01  CERTINFO      USAGE IS POINTER.
    01  REASCODE      PIC 9(8) BINARY.
    01  ERRNO         PIC 9(8) BINARY.
    01  RETCODE       PIC S9(8) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION S HANDSHAKE DNAME
                  SECTYPE  V3CIPHER V3CIPHSEL CERTINFO REASCODE
                  ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'GSKSSOCINIT' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S
ハーフワード 2 進数フィールドであり、セキュア・ソケット接続に使用されるソケットの記述子を指定します。

HANDSHAKE
ハーフワード 2 進数であり、ハンドシェークが実行される方法を指定します。

0 クライアントとして SSL ハンドシェークを実行します (GSK_AS_CLIENT)。

1 サーバーとして SSL ハンドシェークを実行します (GSK_AS_SERVER)。

2 クライアント認証を必要とするサーバーとして SSL ハンドシェークを実行します (GSK_AS_SERVER_WITH_CLIENT_AUTH)。

3 認証なしのクライアントとして SSL ハンドシェークを実行します (GSK_AS_CLIENT_NO_AUTH)。

DNAME
キー・データベース・ファイル内の使用したい項目 (証明書) の識別名またはラベルを表す文字ストリングです。この文字ストリングは、x00 で終わらなければなりません。デフォルトのキー・データベース・ファイル項目を使用する場合は、8 個のブランクからなるストリングを指します。キー・データベース・ファイル項目の識別名は、EZASOKET GETDNBYLAB 関数呼び出しを介して判別できます。

V3CIPHER
SSL 暗号のリストが入っている文字ストリングです。このストリングは、518 ページの『z/OS SSL API』 gsk_get_cipher_info の説明で示されているとおり、16 進値で構成されます。

アプリケーションへ戻されるパラメーター値

SECTYPE
フルワード 2 進数フィールドであり、最小限の受け入れ可能なセキュリティー・プロトコルを示す文字ストリングのアドレスが保管されます。この文字ストリングは、x00 で終わります。有効な値は次のとおりです (二重引用符は除きます)。

- SSL バージョン 3.0 を表す "SSL30"
- TLS バージョン 1.0 を表す "TLS31"

V3CIPHSEL
2 バイトの領域 (アプリケーションによって用意される) であり、このセッション用に選択された SSL バージョン 3.0 暗号仕様に対応する値が保管されます (例: x0009)。

CERTINFO
フルワード 2 進数フィールドであり、クライアントの証明書からの識別名コンポーネントのアドレスが保管されます。このパラメーターが有効なのは、SSL を使用するサーバーに対してクライアント認証が要求されている場合だけです。この領域のレイアウトは、次のとおりです。

4 バイト
Base64 証明書本文へのポインター

4 バイト
Base64 証明書本文の長さ

4 バイト
この接続のセッション ID へのポインター

4 バイト
新しいセッションであるかどうかを示すフラグ

4 バイト
証明書シリアル番号へのポインター

4 バイト
クライアントの共通名へのポインター

4 バイト
市町村へのポインター

4 バイト
都道府県へのポインター

4 バイト
国へのポインター

4 バイト
組織へのポインター

4 バイト
組織部門へのポインター

4 バイト
発行者の共通名へのポインター

4 バイト
発行者の市町村へのポインター

4 バイト
発行者の都道府県へのポインター

4 バイト
発行者の国へのポインター

4 バイト
発行者の組織へのポインター

4 バイト
発行者の組織部門へのポインター

REASCODE
フルワード 2 進数フィールドであり、GSKSSOCINIT 呼び出しの失敗の理由コードが保管されます。値 0 は、関数の正常終了を示します。

ERRNO
フルワード 2 進数フィールド。詳しいエラー情報を示します。

RETCODE
REASCODE が 0 の場合、RETCODE パラメーターには、GSKSOCDATA 構造体へのポインターが入っています。これは、後続の SSL for VSE 操作で使用される必要があります。

GSKSSOCREAD

GSKSSOCREAD 呼び出しは、セキュア・ソケット接続でデータを受信します。

COBOL の例

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION  PIC X(16) VALUE 'GSKSSOCREAD     '.
    01  SSOCDATA      PIC 9(8) BINARY.
    01  NBYTE         PIC 9(8) BINARY.
    01  BUF           PIC X(length of buffer).
    01  ERRNO         PIC 9(8) BINARY.
    01  RETCODE       PIC S9(8) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION SSOCDATA NBYTE BUF
                ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'GSKSSOCREAD' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

SSOCDATA
GSKSSOCINIT 呼び出しによって RETCODE に戻される、GSKSOCDATA 構造体のアドレスです。

NBYTE
フルワード 2 進数であり、BUF のサイズに設定されます。
GSKSSOCREAD は、NBYTE に指定されたバイト数を超えるデータがある場合でも、指定のバイト数のデータだけしか戻しません。データ・バッファーの長さは、64 Kb か、または、受信する最大送信バッファーより少なくとも 32 バイトは大きい値でなければなりません。

BUF 呼び出しの完了時までに埋められるバッファーです。BUF の長さは、最低でも NBYTE の値と同じでなければなりません。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。詳しいエラー情報を示します。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

0 以上
呼び出しは成功しました。RETCODE は、受信したバイト数を表します。

0 より小
エラーが発生しました。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

GSKSSOCRESET

GSKSSOCRESET 呼び出しは、セッションのセキュリティー・パラメーター (暗号鍵など) をリフレッシュします。

COBOL の例

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION  PIC X(16) VALUE 'GSKSSOCRESET    '.
    01  SSOCDATA      PIC 9(8) BINARY.
    01  ERRNO         PIC 9(8) BINARY.
    01  RETCODE       PIC S9(8) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION SSOCDATA
                ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、'GSKSSOCRESET' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

SSOCDATA

GSKSSOCINIT 呼び出しによって RETCODE に戻される、GSKSOCDATA 構造体のアドレスです。

アプリケーションへ戻されるパラメーター値

ERRNO

フルワード 2 進数フィールド。詳しいエラー情報を示します。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

0　　呼び出しは成功しました。

0 より小

エラーが発生しました。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

GSKSSOCWRITE

GSKSSOCWRITE 呼び出しは、セキュア・ソケット接続でデータを送信します。

COBOL の例

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION  PIC X(16) VALUE 'GSKSSOCWRITE    '.
    01  SSOCDATA      PIC 9(8) BINARY.
    01  NBYTE         PIC 9(8) BINARY.
    01  BUF           PIC X(length of buffer).
    01  ERRNO         PIC 9(8) BINARY.
    01  RETCODE       PIC S9(8) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION SSOCDATA NBYTE BUF
                ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'GSKSSOCWRITE' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

SSOCDATA
GSKSSOCINIT 呼び出しによって RETCODE に戻される、GSKSOCDATA 構造体のアドレスです。

NBYTE
フルワード 2 進数であり、送信するバイト数に設定されます。サポートされる最大バイト数は 64K です。

BUF 送信されるデータが入っているバッファーを指定します。BUF のサイズは、NBYTE に指定されているサイズであるべきです。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。詳しいエラー情報を示します。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

0 以上
呼び出しは成功しました。RETCODE は、送信されたバイト数を表します。

0 より小
エラーが発生しました。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

GSKUNINIT

GSKUNINIT 呼び出しは、SSL 環境に関する現行の設定全体を除去します。

セッションのタイムアウト値や SSL プロトコルなどのフィールドが除去されます。

COBOL の例

```
WORKING-STORAGE SECTION.
    01  SOC-FUNCTION  PIC X(16) VALUE 'GSKUNINIT       '.
    01  ERRNO         PIC 9(8) BINARY.
    01  RETCODE       PIC S9(8) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-FUNCTION
                ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'GSKUNINIT' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。詳しいエラー情報を示します。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

0 呼び出しは成功しました。

0 以外
エラーが発生しました。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

INITAPI

INITAPI 呼び出しは、アプリケーションを TCP/IP インターフェースに接続します。

COBOL、PL/I、またはアセンブラー言語で書かれたほとんどすべてのソケット・プログラムは、他のソケット・マクロを発行する前に INITAPI マクロを発行する必要があります。

この規則の例外は、以下の呼び出しで、これらが最初に発行されると、デフォルトの INITAPI 呼び出しが生成されます。

- GETCLIENTID
- GETHOSTID
- GETHOSTNAME
- SELECT
- SELECTEX
- SOCKET
- TAKESOCKET

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'INITAPI'.
    01  MAXSOC          PIC 9(4) BINARY.
    01  IDENT.
        02  TCPNAME     PIC X(8).
        02  ADSNAME     PIC X(8).
    01  SUBTASK         PIC X(8).
    01  MAXSNO          PIC 9(8) BINARY.
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC IDENT SUBTASK
     MAXSNO ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、INITAPI を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

MAXSOC

入力パラメーター。ハーフワード 2 進数フィールドであり、このアプリケーションに対してサポートされる最大ソケット数を指定します。現在、TCP/IP for z/VSE はこの入力を無視し、サポートされる最大ソケット数をデフォルトの 8192 に設定します。ソケット記述子の数値の範囲は、0 から 8191 です。

IDENT

TCP/IP アドレス・スペースと呼び出し側プログラムのアドレス・スペースの ID を含む構造体です。アドレス・スペースからの INITAPI 呼び出しに IDENT を指定します。

TCPNAME

z/VSE 4.2 以降、このパラメーターを、このアプリケーションで使用するローカル TCP/IP スタックの選択に使用できるようになりました。この 8 バイトのパラメーターは、「SOCKETnn」または単に「nn」と設定できます (左寄せまたは右寄せし、6 つのブランクを埋め込む)。値「nn」で、TCP/IP スタートアップ JCL の ID パラメーターで指定された、選択された TCP/IP スタックの ID を判別します。

ADSNAME

このパラメーターは、EZA 処理環境で使用される TCP/IP インターフェース・ルーチンの名前の指定に使用できます。ここに何も指定しないと、IBM 提供の TCP/IP インターフェース・ルーチン EZASOH99 が使用されます。この指定は、JCL ステートメント // SETPARM [SYSTEM,] EZA$PHA='routine-name' で上書きできることに注意してください。

注**:** LE/C ソケット API マルチプレクサーを使用して、アプリケーションで使用する TCP/IP および SSL の実装を選択する方法については、 95 ページの『第 10 章 使用する TCP/IP と SSL の実装を選択』を参照してください。

SUBTASK

8 バイトのフィールドであり、1 つのアドレス・スペース内の複数のサブタスクを識別するのに使用される固有のサブタスク ID を含みます。独自のジョブ名をサブタスク名の一部として使用してください。そうすると、複数の INITAPI コマンドを同じアドレス・スペースから発行する場合、各 SUBTASK パラメーターが固有になります。これが指定されないか、8 個のブランクが指定された場合、デフォルトのサブタスク名が使用されます。バッチ環境では、次のようになります。

バイト **0-2**

ジョブ名の先頭 3 文字

バイト **3**
hex F0

バイト **4-7**
VSE タスク ID

CICS トランザクション環境では、次のようになります。

バイト **0-2**
CICS EIBTRNID (トランザクション ID)

バイト **3**
hex F1

バイト **4-7**
CICS EIBTASKN (タスク番号)

アプリケーションへ戻されるパラメーター値

MAXSNO
出力パラメーター。フルワード 2 進数フィールドであり、このアプリケーションに割り当てることのできる最大の記述子値を含みます。アプリケーションに割り当てられたソケット記述子は連続した順序になりません。現在、TCP/IP for z/VSE は常に 8191 を戻します。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

IOCTL

IOCTL 呼び出しは、ソケットの特定の動作特性を制御することができます。

IOCTL マクロを発行する前に、制御する特性を表す値を COMMAND フィールドにロードしなければなりません。

可変長のパラメーター REQARG および RETARG が、IOCTL との間でやり取りされる引数です。REQARG と RETARG の長さは、COMMAND に指定する値によって決まります。

COBOL の例

```
WORKING-STORAGE SECTION.
01  SOKET-FUNCTION         PIC X(16) VALUE 'IOCTL'.
01  S                       PIC 9(4)  BINARY.
01  COMMAND                 PIC 9(4)  BINARY.

01  IFREQ,
  3 NAME                    PIC X(16).
```

```
   3 FAMILY                PIC 9(4)  BINARY.
   3 PORT                  PIC 9(4)  BINARY.
   3 ADDRESS               PIC 9(8)  BINARY.
   3 RESERVED              PIC X(8).

01  IFREQOUT,
   3 NAME                  PIC X(16).
   3 FAMILY                PIC 9(4)  BINARY.
   3 PORT                  PIC 9(4)  BINARY.
   3 ADDRESS               PIC 9(8)  BINARY.
   3 RESERVED              PIC X(8).

01  GRP_IOCTL_TABLE(100)
 02 IOCTL_ENTRY,
   3 NAME                  PIC X(16).
   3 FAMILY                PIC 9(4)  BINARY.
   3 PORT                  PIC 9(4)  BINARY.
   3 ADDRESS               PIC 9(8)  BINARY.
   3 NULLS                 PIC X(8).

01  IOCTL_REQARG           POINTER ;
01  IOCTL_RETARG           POINTER ;
01  ERRNO                  PIC 9(8) BINARY.
01  RETCODE                PIC 9(8) BINARY.


PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S COMMAND REQARG
           RETARG ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、IOCTL を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S
ハーフワード 2 進数であり、制御対象となるソケットのソケット記述子に設定されます。

COMMAND
動作特性を制御するため、このフィールドは 400 ページの表 9 に示す値に設定されます。

REQARG および **RETARG**
REQARG は、引数を IOCTL に渡すために使用され、RETARG は引数を受け取ります。REQARG と RETARG の長さと意味については、 400 ページの表 9 を参照してください。

アプリケーションへ戻されるパラメーター値

RETARG
COMMAND の値に基づいたサイズの配列を戻します。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

LISTEN

LISTEN 呼び出しは、着信する接続要求用に、指定された長さの接続要求待ち行列を作成します。

重要: LISTEN 呼び出しは、データグラム・ソケットまたはロー・ソケットにはサポートされていません。

LISTEN 呼び出しは、一般的に、クライアントからの接続要求を受信するためにサーバーが使用します。接続要求が受信されると、後続の ACCEPT 呼び出しによって新規ソケットが作成され、元のソケットは以降の接続要求の listen を続行します。LISTEN 呼び出しは、能動ソケットを受動ソケットに変換し、そのソケットがクライアントからの接続要求を受け入れる状態にします。いったん受動になったソケットは、接続要求を開始できません。LISTEN 呼び出しでは、事前に BIND 要求を発行しておく必要があります。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'LISTEN'.
    01  S               PIC 9(4) BINARY.
    01  BACKLOG         PIC 9(8) BINARY.
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
    CALL 'EZASOKET' USING SOC-FUNCTION S BACKLOG ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、LISTEN を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S ハーフワード 2 進数であり、ソケット記述子に設定されます。

BACKLOG
フルワード 2 進数であり、待ち行列に入れられる通信要求の数に設定されます。このパラメーターは無視されます。常に値 1 が想定されます。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

NTOP

NTOP 呼び出しは、IP アドレスをその数値バイナリー形式から、標準テキスト表示形式に変換します。

正常に完了した場合、NTOP は変換された IP アドレスを、指定されたバッファーに返します。

重要: この関数呼び出しは TCP/IP for z/VSE では使用できません。

COBOL の例

```
WORKING-STORAGE SECTION.
    01 SOC-ACCEPT-FUNCTION PIC X(16) VALUE IS 'ACCEPT'.
    01 SOC-NTOP-FUNCTION PIC X(16) VALUE IS 'NTOP'.
    01 S PIC 9(4) BINARY.

* IPv4 socket structure.
    01 NAME.
       03 FAMILY PIC 9(4) BINARY.
       03 PORT PIC 9(4) BINARY.
       03 IP-ADDRESS PIC 9(8) BINARY.
       03 RESERVED PIC X(8).

* IPv6 socket structure.
    01 NAME.
       03 FAMILY PIC 9(4) BINARY.
       03 PORT PIC 9(4) BINARY.
       03 FLOWINFO PIC 9(8) BINARY.
       03 IP-ADDRESS.
          10 FILLER PIC 9(16) BINARY.
          10 FILLER PIC 9(16) BINARY.
       03 SCOPE-ID PIC 9(8) BINARY.
    01 NTOP-FAMILY PIC 9(8) BINARY.
    01 ERRNO PIC 9(8) BINARY.
    01 RETCODE PIC S9(8) BINARY.
    01 PRESENTABLE-ADDRESS PIC X(45).
    01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY.

PROCEDURE DIVISION.
    CALL 'EZASOKET' USING SOC-ACCEPT-FUNCTION S NAME ERRNO RETCODE.
    CALL 'EZASOKET' USING SOC-NTOP-FUNCTION NTOP-FAMILY IP-ADDRESS
          PRESENTABLE-ADDRESS PRESENTABLE-ADDRESS-LEN ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-NTOP-FUNCTION

16 バイトの文字フィールドであり、NTOP を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

NTOP-FAMILY
変換される IP アドレスのアドレス・ファミリー。10 進数 2 の値を AF_INET に、19 の値を AF_INET6 に指定しなければなりません。

IP-ADDRESS
変換された数値バイナリー形式の IPv4 または IPv6 アドレスが入るフィールド。IPv4 アドレスの場合はこのフィールドはフルワードでなければならず、IPv6 アドレスの場合はこのフィールドは 16 バイトでなければなりません。このアドレスは、ネットワーク・バイト・オーダーでなければなりません。

アプリケーションへ戻されるパラメーター値

PRESENTABLE-ADDRESS
変換された標準テキスト表示形式の IPv4 または IPv6 アドレスを受け取るために使用するフィールド。IPv4 の場合はアドレスはドット 10 進形式で、IPv6 の場合はアドレスはコロン 16 進形式です。IPv4 アドレスのサイズは最大で 15 バイトであり、変換された IPv6 アドレスのサイズは最大で 45 バイトです。PRESENTABLE-ADDRESS の値の実際の長さについては、PRESENTABLE-ADDRESS-LEN で返された値を参照してください。

PRESENTABLE-ADDRESS-LEN
最初は入力パラメーター。ハーフワード 2 進数フィールドのアドレスであり、入力で PRESENTABLE-ADDRESS フィールドの長さを指定するために使用され、成功して戻ると、変換された IP アドレスの長さが入ります。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。そうでない場合には、ERRNO フィールドは無視できます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

PTON

PTON 呼び出しは、IP アドレスをその標準テキスト表示形式から数値バイナリー形式に変換します。

正常に完了した場合、PTON は変換された IP アドレスを、指定されたバッファーに返します。

重要: この関数呼び出しは TCP/IP for z/VSE では使用できません。

COBOL の例

```
WORKING-STORAGE SECTION.
    01 SOC-BIND-FUNCTION PIC X(16) VALUE IS 'BIND'.
    01 SOC-PTON-FUNCTION PIC X(16) VALUE IS 'PTON'.
    01 S PIC 9(4) BINARY.
```

```
* IPv4 socket structure.
    01 NAME.
       03 FAMILY PIC 9(4) BINARY.
       03 PORT PIC 9(4) BINARY.
       03 IP-ADDRESS PIC 9(8) BINARY.
       03 RESERVED PIC X(8).

* IPv6 socket structure.
    01 NAME.
       03 FAMILY PIC 9(4) BINARY.
       03 PORT PIC 9(4) BINARY.
       03 FLOWINFO PIC 9(8) BINARY.
       03 IP-ADDRESS.
          10 FILLER PIC 9(16) BINARY.
          10 FILLER PIC 9(16) BINARY.
       03 SCOPE-ID PIC 9(8) BINARY.
    01 AF-INET PIC 9(8) BINARY VALUE 2.
    01 AF-INET6 PIC 9(8) BINARY VALUE 19.

* IPv4 address.
    01 PRESENTABLE-ADDRESS PIC X(45).
    01 PRESENTABLE-ADDRESS-IPV4 REDEFINES PRESENTABLE-ADDRESS.
       05 PRESENTABLE-IPV4-ADDRESS PIC X(15) VALUE '192.26.5.19'.
       05 FILLER PIC X(30).
    01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY VALUE 11.

* IPv6 address.
    01 PRESENTABLE-ADDRESS PIC X(45) VALUE '12f9:0:0:c30:123:457:9cb:1112'.
    01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY VALUE 29.

* IPv4-mapped IPv6 address.
    01 PRESENTABLE-ADDRESS PIC X(45) VALUE '12f9:0:0:c30:123:457:192.26.5.19'.
    01 PRESENTABLE-ADDRESS-LEN PIC 9(4) BINARY VALUE 32.
    01 ERRNO PIC 9(8) BINARY.
    01 RETCODE PIC S9(8) BINARY.

PROCEDURE DIVISION.
* IPv4 address.
    CALL 'EZASOKET' USING SOC-PTON-FUNCTION AF-INET PRESENTABLE-ADDRESS
    PRESENTABLE-ADDRESS-LEN IP-ADDRESS ERRNO RETURN-CODE.

* IPv6 address.
    CALL 'EZASOKET' USING SOC-PTON-FUNCTION AF-INET6 PRESENTABLE-ADDRESS
                    PRESENTABLE-ADDRESS-LEN IP-ADDRESS ERRNO RETURN-CODE.
    CALL 'EZASOKET' USING SOC-BIND-FUNCTION S NAME ERRNO RETURN-CODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-PTON-FUNCTION

16 バイトの文字フィールドであり、PTON を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

AF-INET6

変換される IP アドレスのアドレス・ファミリー。10 進数 2 の値を AF_INET に、19 の値を AF_INET6 に指定しなければなりません。

PRESENTABLE-ADDRESS

変換された標準テキスト表示形式の IPv4 または IPv6 アドレスが入るフィールド。IPv4 の場合はアドレスはドット 10 進形式で、IPv6 の場合はアドレスはコロン 16 進形式です。

PRESENTABLE-ADDRESS-LEN
ハーフワード 2 進数のアドレスのフィールドであり、変換される IP アドレスの長さが入らなければなりません。

アプリケーションへ戻されるパラメーター値

IP-ADDRESS
変換された数値バイナリー形式の IPv4 または IPv6 アドレスが入るフィールド。IPv4 アドレスの場合はこのフィールドはフルワードでなければならず、IPv6 アドレスの場合はこのフィールドは 16 バイトでなければなりません。このアドレスは、ネットワーク・バイト・オーダーでなければなりません。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。そうでない場合には、ERRNO フィールドは無視できます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

READ

READ 呼び出しでは、ソケットでデータを読み取ります。

これは、標準的な TCP/IP データ読み取り操作です。データグラム・パケットが長すぎて、提供されたバッファーに収まらない場合には、データグラム・ソケットは、余分なバイトを廃棄します。

ストリーム・ソケットの場合、データは、データを分離する境界のない情報ストリームとして処理されます。例えば、プログラム A とプログラム B がストリーム・ソケットで接続されていて、プログラム A が 1000 バイトを送信する場合、この関数の呼び出しは、1000 バイト全部までの任意のバイト数を戻すことができます。戻されたバイト数は RETCODE に入ります。従って、ストリーム・ソケットを使用するプログラムでは、この関数をループに入れ、全データが受信されるまで繰り返す必要があります。

注: ASCII 入力データを EBCDIC に変換するサブルーチンについては、 324 ページの『EZACIC05』を参照してください。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'READ'.
    01  S               PIC 9(4) BINARY.
    01  NBYTE           PIC 9(8) BINARY.
    01  BUF             PIC X(length of buffer).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.
```

```
    PROCEDURE
         CALL 'EZASOKET' USING SOC-FUNCTION S NBYTE BUF
                               ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、READ を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S ハーフワード 2 進数であり、データを読み取るソケットのソケット記述子に設定されます。

NBYTE
フルワード 2 進数であり、BUF のサイズに設定されます。READ は、NBYTE に指定されたバイト数を超えるデータがある場合でも、指定のバイト数のデータだけしか戻しません。

アプリケーションへ戻されるパラメーター値

BUF 呼び出しの完了によって埋められる入力バッファーです。BUF の長さは、最低でも NBYTE の値と同じでなければなりません。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	戻りコード 0 は、接続がクローズしていて、使用可能なデータがないことを意味します。
>0	正の値は、バッファーにコピーされたバイト数を示します。
-1	ERRNO のエラー・コードをチェックしてください。

READV

READV 呼び出しでは、ソケットでデータを読み取り、そのデータをバッファー・セットに保管します。

データグラム・ソケットが長すぎて、提供されたバッファーに収まらない場合、余分なバイトは廃棄されます。

COBOL の例

```
WORKING-STORAGE SECTION.
       01 SOC-FUNCTION        PIC X(16) VALUE 'READV'.
       01 S                   PIC 9(4) BINARY.
       01 ERRNO               PIC 9(8) BINARY.
       01 RETCODE             PIC S9(8) BINARY.
       01 IOVCNT              PIC 9(8) BINARY.
```

```
       01 IOV.
          03 BUFFER-ENTRY OCCURS 5 TIMES.
             05 IOV-POINTER       USAGE IS POINTER.
             05 RESERVED          PIC X(4).
             05 IOV-LENGTH        PIC 9(8) BINARY.
       01 HEAPID              PIC S9(9) BINARY VALUE IS 0.

    PROCEDURE DIVISION.


      MOVE 50 TO IOV-LENGTH(1).
      MOVE 50 TO IOV-LENGTH(2).
      MOVE 50 TO IOV-LENGTH(3).
      MOVE 50 TO IOV-LENGTH(4).
      MOVE 50 TO IOV-LENGTH(5).
      CALL 'CEEGTST' USING HEAPID, IOV-LENGTH(1), IOV-POINTER(1),
                           FC.
      CALL 'CEEGTST' USING HEAPID, IOV-LENGTH(2), IOV-POINTER(2),
                           FC.
      CALL 'CEEGTST' USING HEAPID, IOV-LENGTH(3), IOV-POINTER(3),
                           FC.
      CALL 'CEEGTST' USING HEAPID, IOV-LENGTH(4), IOV-POINTER(4),
                           FC.
      CALL 'CEEGTST' USING HEAPID, IOV-LENGTH(5), IOV-POINTER(5),
                           FC.

      CALL 'EZASOKET' USING SOC-FUNCTION S IOV IOVCNT
                    ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、READV を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S ハーフワード 2 進数であり、データを読み取るソケットのソケット記述子に設定されます。

IOV IOVCNT の値と等しい数の構造体の 3 つのフルワード構造の配列。

構造体の形式は、次のとおりです。

- フルワード 1: データ・バッファーのアドレス。このバッファーは、呼び出しの完了によって埋められます。
- フルワード 2: 予約済み
- フルワード 3: フルワード 1 で参照されるデータ・バッファーの長さ

IOVCNT
フルワードの 2 進数フィールドであり、この呼び出しに提供されるデータ・バッファー数を指定します。最大は 120 です。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の値の場合、このフィールドの中にはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	接続はクローズしていて、データは使用できません。
>0	バッファーにコピーされたバイト数。
-1	エラーが発生しました。ERRNO のエラー・コードをチェックしてください。

RECV

RECV 呼び出しは、READ と同様に、記述子 S のソケットでデータを受信します。

RECV は、接続されたソケットにのみ適用されます。データグラム・パケットが長すぎて、提供されたバッファーに収まらない場合には、データグラム・ソケットは、余分なバイトを廃棄します。

ストリーム・ソケットの場合、データは、データを分離する境界のない情報ストリームとして処理されます。例えば、プログラム A とプログラム B がストリーム・ソケットで接続されていて、プログラム A が 1000 バイトを送信する場合、この関数の呼び出しは、1000 バイト全部までの任意のバイト数を戻すことができます。戻されたバイト数は RETCODE に入ります。従って、ストリーム・ソケットを使用するプログラムでは、RECV をループに入れ、全データが受信されるまで繰り返す必要があります。

ソケットでデータを使用できず、ソケットがブロッキング・モードの場合、RECV はデータが到着するまで呼び出し元をブロックします。データが使用可能でなく、ソケットが非ブロッキング・モードの場合、RECV は -1 を戻し、ERRNO EWOULDBLOCK を設定します。非ブロッキング・モードの設定方法の説明については、 242 ページの『FCNTL』または 287 ページの『IOCTL』を参照してください。

注: ASCII 入力データを EBCDIC に変換するサブルーチンについては、 324 ページの『EZACIC05』を参照してください。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'RECV'.
    01  S               PIC 9(4) BINARY.
    01  FLAGS           PIC 9(8) BINARY.
        88 NO-FLAG                  VALUE IS 0
    01  NBYTE           PIC 9(8) BINARY.
    01  BUF             PIC X(length of buffer).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE BUF
                     ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、RECV を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S ハーフワード 2 進数であり、データを受信するソケットのソケット記述子に設定されます。

FLAGS
フルワード 2 進数フィールドであり、NO-FLAG または 0 に設定されなければなりません。

NBYTE
BUF のサイズに設定されるフルワード 2 進数のアドレス、または値。
RECV は、NBYTE に指定されたバイト数を超えるデータがある場合でも、指定のバイト数のデータだけしか受信しません。

アプリケーションへ戻されるパラメーター値

BUF データを受け取る入力バッファーです。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	ソケットがクローズされた
>0	正の戻りコードは、バッファーにコピーされたバイト数を示します。
-1	ERRNO のエラー・コードをチェックしてください。

RECVFROM

RECVFROM 呼び出しは、記述子 S のソケットでデータを受信し、そのデータをバッファーに保管します。

RECVFROM 呼び出しは、接続済みソケットと未接続ソケットの両方に適用されます。ソケット・アドレスが NAME 構造体に戻されます。データグラム・パケットが長すぎて、提供されたバッファーに収まらない場合には、データグラム・ソケットは、余分なバイトを廃棄します。

データグラム・プロトコルの場合、recvfrom() が、各着信データグラムに関連するソース・アドレスを戻します。TCP のようなコネクション指向プロトコルの場合、getpeername() が、接続の他の終端に関連するアドレスを戻します。

NAME がゼロ以外の場合は、この呼び出しは送信側のアドレスを戻します。
NBYTE パラメーターは、バッファーのサイズに設定される必要があります。

戻り時に、NBYTE には受信したデータのバイト数が入ります。

ストリーム・ソケットの場合、データは、データを分離する境界のない情報ストリームとして処理されます。例えば、プログラム A とプログラム B がストリーム・ソケットで接続されていて、プログラム A が 1000 バイトを送信する場合、この関数の呼び出しは、1000 バイト全部までの任意のバイト数を戻すことができます。戻されたバイト数は RETCODE に入ります。従って、ストリーム・ソケットを使用するプログラムでは、RECVFROM をループに入れ、全データが受信されるまで繰り返す必要があります。

ソケットでデータが使用できず、ソケットがブロッキング・モードの場合、RECVFROM はデータが到着するまで呼び出し元をブロックします。データが使用可能でなく、ソケットが非ブロッキング・モードの場合、RECVFROM は -1 を戻し、ERRNO EWOULDBLOCK を設定します。非ブロッキング・モードの設定方法の説明については、 242 ページの『FCNTL』または 287 ページの『IOCTL』を参照してください。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

注**:** ASCII 入力データを EBCDIC に変換するサブルーチンについては、 324 ページの『EZACIC05』を参照してください。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'RECVFROM'.
    01  S               PIC 9(4) BINARY.
    01  FLAGS           PIC 9(8) BINARY.
        88  NO-FLAG                VALUE IS 0.
    01  NBYTE           PIC 9(8) BINARY.
    01  BUF             PIC X(length of buffer).
    01  NAME.
        03  FAMILY      PIC 9(4) BINARY.
        03  PORT        PIC 9(4) BINARY.
        03  IP-ADDRESS  PIC 9(8) BINARY.
        03  RESERVED    PIC X(8).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS
                    NBYTE BUF NAME ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、RECVFROM を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S
ハーフワード 2 進数であり、データを受信するソケットのソケット記述子に設定されます。

FLAGS
フルワード 2 進数フィールドであり、NO-FLAG または 0 に設定されなければなりません。

NBYTE
フルワード 2 進数であり、入力バッファーの長さを指定します。

アプリケーションへ戻されるパラメーター値

BUF 入力データを受け取る入力バッファーを定義します。

NAME
最初は、IPv4 または IPv6 アプリケーションが構造体へのポインターを設定し、呼び出しの完了時に、その構造体にピア・ソケット名が入ります。NAME パラメーター値がゼロ以外の場合は、メッセージの IPv4 または IPv6 ソース・アドレスが挿入されます。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
説明

FAMILY
ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、送信側ソケットのポート番号を指定します。

IPv4-ADDRESS
フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
説明

NAMELEN
IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、送信側ソケットのポート番号を指定します。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	ソケットがクローズされた。
>0	正の戻りコードは、読み取り呼び出しで転送されたデータのバイト数を示します。
-1	ERRNO のエラー・コードをチェックしてください。

SELECT

複数の入出力操作が発生するプロセスにおいて、プログラムは 1 つまたは複数の操作の完了を待機できる必要があります。

例えば、ブロッキング・モードが設定されている複数のソケットに READ を発行するプログラムを想定します。ソケットは READ 呼び出しでブロックするので、一度に 1 つのソケットを読みとることしかできません。ソケットを非ブロッキングに設定すると、この問題は解決できますが、データが使用可能になるまで各ソケットを繰り返しポーリングすることが必要になります。SELECT 呼び出しを使用する

と、いくつかのソケットをテストして、そのうちの 1 つが作動可能である場合にのみ、その後の入出力呼び出しを実行するようにできます。これにより、入出力呼び出しがブロックしないようにできます。

プログラム内でタイマーとして SELECT 呼び出しを使用するには、以下のいずれかを行います。

- 読み取り、書き込み、および例外の各配列をゼロに設定する。
- MAXSOC <= 0 を設定する。

テストするソケットの定義

SELECT 呼び出しは、読み取り操作、書き込み操作、および例外操作についてモニターします。

- ソケットが読み取り可能なのは、以下のいずれかが発生した場合です。
  - 指定されたソケット用のバッファーに入力データが含まれているかどうか。あるソケットに対する入力データがある場合、そのソケットでの読み取り操作はブロックしません。
  - そのソケットで接続が要求された。
- ソケットが書き込み可能なのは、TCP/IP が追加の出力データに対応できる場合です。あるソケットに対して TCP/IP が追加の出力を受け入れ可能である場合、そのソケットでの書き込み操作はブロックしません。
- 指定されたソケットで例外条件が発生した場合、それは、そのソケットで TAKESOCKET が発生したことを示します。

各ソケット記述子は、ビット・ストリング中の 1 つのビットで表されます。ビット・ストリングは、32 ビットのフルワードに含まれ、右から左に番号が付与されます。右端のビットがソケット記述子 0 を表し、左端のビットがソケット記述子 31 を表します。プロセスが 32 個以下のソケットを使用する場合、ビット・ストリングは 1 個のフルワードです。プロセスが 33 個のソケットを使用する場合、ビット・ストリングは 2 個のフルワードです。最初のフルワードがソケット記述子 0 から 31 を表し、2 番目のフルワードがソケット記述子 32 から 63 を表します。ストリング内のビットをオンにすることで、テスト対象のソケットを定義します。

注: プログラム EZACIC06 を使用してストリング内の各ビットを文字に変換すると、COBOL でのストリング処理を簡単にできます。詳しくは、325 ページの『EZACIC06』を参照してください。

読み取り操作

読み取り操作には、ACCEPT、READ、RECV、RECVFROM 呼び出しがあります。ソケットに対するデータが受信されたときか、接続要求が発生したとき、そのソケットは読み取り可能です。

いくつかのソケットのうちのどれかが読み取り可能かどうかをテストするには、RSNDMSK 中の対応するビットを 1 に設定してから、SELECT 呼び出しを発行します。SELECT 呼び出しが戻ると、RRETMSK 中の対応するビットによって、ソケットが読み取り可能かどうかが示されます。

書き込み操作

ソケットが書き込みに対して選択されている (書き込み可能である) ということになるのは、以下の場合です。

- TCP/IP は、追加の発信データを受け入れることができる。
- ソケットは非ブロッキングとマークされていて、前の CONNECT が即時に完了しなかった。この場合、CONNECT が戻した ERRNO の値は EINPROGRESS です。CONNECT が完了したら、このソケットが選択され、書き込みできる状態になります。

WRITE、SEND、または SENDTO の呼び出しは、送信されるデータ量が TCP/IP が受け入れ可能な量を超えると、ブロックします。これを回避するには、書き込み操作の前に SELECT 呼び出しを使用して、ソケットが確実に書き込み可能な状態にします。

いくつかのソケットのどれかが書き込み可能かどうかをテストするには、WSNDMSK 中のそれらのソケットに対応するビットを 1 に設定してから、SELECT 呼び出しを発行します。SELECT 呼び出しが戻ると、WRETMSK 中の対応するビットによって、ソケットが書き込み可能かどうかが示されます。

例外操作

テストする各ソケットについて、SELECT 呼び出しは、例外条件があるかどうかをチェックできます。次の 2 つの例外条件がサポートされています。

- 呼び出し側プログラム (並行サーバー) が GIVESOCKET コマンドを発行し、ターゲットの子サーバーが正常に TAKESOCKET 呼び出しを発行した。この条件が選択されている場合、呼び出し側プログラム (並行サーバー) は、CLOSE コマンドを発行して自身をソケットから切り離すべきです。
- ソケットがアウト・オブ・バンドのデータを受信した。この条件が発生した場合、READ は、アウト・オブ・バンドのデータをプログラム・データの前に戻します。

いくつかのソケットのどれかに例外条件があるかどうかをテストするには、それらのソケットを表す ESNDMSK ビットを 1 に設定します。SELECT 呼び出しが戻ると、ERETMSK 中の対応するビットによって、例外条件のあるソケットが示されます。

MAXSOC パラメーター

SELECT 呼び出しは、結果を戻す前に、各ストリング中の各ビットをテストしなければなりません。効率をよくするため、MAXSOC パラメーターを使用して、任意のイベント・タイプのテストが必要なソケット記述子の最大値を指定することができます。SELECT 呼び出しは、ゼロから MAXSOC 値までの範囲のビットだけをテストします。

TIMEOUT パラメーター

TIMEOUT パラメーターに指定した時間が経過してもイベントが何も検出されない場合、SELECT 呼び出しは、RETCODE を 0 に設定して戻ります。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SELECT'.
    01  MAXSOC          PIC 9(8) BINARY.
    01  TIMEOUT.
        03  TIMEOUT-SECONDS   PIC 9(8) BINARY.
        03  TIMEOUT-MICROSEC  PIC 9(8) BINARY.
    01  RSNDMSK        PIC X(*).
    01  WSNDMSK        PIC X(*).
    01  ESNDMSK        PIC X(*).
    01  RRETMSK        PIC X(*).
    01  WRETMSK        PIC X(*).
    01  ERETMSK        PIC X(*).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
                    RSNDMSK WSNDMSK ESNDMSK
                    RRETMSK WRETMSK ERETMSK
                    ERRNO RETCODE.
```

以下の例は、SELECT 呼び出し命令を示しています。

* ビット・マスクの長さは、次の式で決定されます。

```
((maximum socket number +32)/32 (drop the remainder))*4
```

ビット・マスクは、32 ビットのフルワードであり、各ソケットが 1 ビットに対応します。32 個までのソケットは、1 つの 32 ビットのマスク [PIC X(4)] で表されます。33 個のソケットがある場合、2 つの 32 ビットのマスク [PIC X(8)] を割り振る必要があります。

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION

16 バイトの文字フィールドであり、SELECT を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

MAXSOC

入力パラメーター。フルワード 2 進数フィールドであり、チェック対象のソケット記述子の最大値に 1 を加えた数値を指定します (TCP/IP for z/VSE は 0 から 8191 までの値のソケット記述子をサポートするためです)。

TIMEOUT

TIMEOUT が正の値の場合、その値は、選択が最大どれだけの間隔だけ待ってから完了するのかを示します。TIMEOUT-SECONDS が負の値の場合、SELECT 呼び出しは、ソケットが作動可能になるまでブロックします。ソケットをポーリングし、即時に戻るためには、TIMEOUT 値をゼロに指定します。

TIMEOUT は、次の 2 ワードの TIMEOUT で指定されます。

- TIMEOUT-SECONDS は、TIMEOUT フィールドの 1 番目のワードであり、タイムアウト値の秒の部分を表すコンポーネントです。

- TIMEOUT-MICROSEC は、TIMEOUT フィールドの 2 番目のワードであり、タイムアウト値のマイクロ秒の部分を表すコンポーネントです (0—999999)。

例えば、SELECT が 3.5 秒でタイムアウトになるようにするには、TIMEOUT-SECONDS を 3 に、TIMEOUT-MICROSEC を 500000 に設定します。

RSNDMSK

読み取りイベント状況を要求するために送信されるビット・ストリングです。

- 保留中の読み取りイベントをチェックする各ソケットごとに、このストリング中の対応するビットが 1 に設定される必要があります。
- 無視するソケットについては、対応するビットの値は 0 に設定される必要があります。

このパラメーターがすべてゼロに設定されている場合、SELECT は読み取りイベントをチェックしません。

WSNDMSK

書き込みイベント状況を要求するために送信されるビット・ストリングです。

- 保留中の書き込みイベントをチェックする各ソケットごとに、このストリング中の対応するビットが 1 に設定される必要があります。
- 無視するソケットについては、対応するビットの値は 0 に設定される必要があります。

このパラメーターがすべてゼロに設定されている場合、SELECT は書き込みイベントをチェックしません。

ESNDMSK

例外イベント状況を要求するために送信されるビット・ストリングです。

- 保留中の例外イベントをチェックする各ソケットごとに、このストリング中の対応するビットが 1 に設定される必要があります。
- 無視するソケットについては、対応するビットは 0 に設定される必要があります。

このパラメーターがすべてゼロに設定されている場合、SELECT は例外イベントをチェックしません。

アプリケーションへ戻されるパラメーター値

RRETMSK

読み取りイベントの状況が戻されるビット・ストリングです。このストリングの長さは、チェック対象のソケットの最大数と同じである必要があります。読み取り可能な各ソケットについては、このストリング中の対応するビットが 1 に設定されます。読み取り可能でないソケットを表すビットは、0 に設定されます。

WRETMSK

書き込みイベントの状況が戻されるビット・ストリングです。このストリングの長さは、チェック対象のソケットの最大数と同じである必要がありま

す。書き込み可能な各ソケットについては、このストリング中の対応するビットが 1 に設定されます。書き込み可能でないソケットを表すビットは、0 に設定されます。

ERETMSK
例外イベントの状況が戻されるビット・ストリングです。このストリングの長さは、チェック対象のソケットの最大数と同じである必要があります。例外状況を持つ各ソケットについては、このストリング中の対応するビットが 1 に設定されます。例外状況を持たないソケットを表すビットは、0 に設定されます。

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
>0	3 つのマスク内のすべての作動可能ソケットの合計を示します。
0	SELECT 制限時間が満了になったことを示します。
-1	ERRNO のエラー・コードをチェックしてください。

SELECTEX

SELECTEX 呼び出しは、ソケットの集合、タイム値、1 つの ECB または ECB リストをモニターします。

この呼び出しが完了するのは、ソケットの 1 つにアクティビティーがあるとき、タイム値が満了したとき、または、ECB の 1 つがポストされたときのいずれかです。

プログラム内でタイマーとして SELECTEX 呼び出しを使用するには、以下のいずれかを行います。

- 読み取り、書き込み、および例外の各配列をゼロに設定する。
- MAXSOC <= 0 を設定する。

ソケットのテストについての詳しい説明は、 300 ページの『SELECT』を参照してください。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SELECTEX'.
    01  MAXSOC          PIC 9(8)   BINARY.
    01  TIMEOUT.
        03  TIMEOUT-SECONDS    PIC 9(8) BINARY.
        03  TIMEOUT-MINUTES    PIC 9(8) BINARY.
    01  RSNDMSK        PIC X(*).
    01  WSNDMSK        PIC X(*).
    01  ESNDMSK        PIC X(*).
    01  RRETMSK        PIC X(*).
    01  WRETMSK        PIC X(*).
    01  ERETMSK        PIC X(*).
```

```
01  SELECB          PIC X(4).
01  ERRNO           PIC 9(8)   BINARY.
01  RETCODE         PIC S9(8)  BINARY.
```

ここで、* は選択されるマスクのサイズです

```
PROCEDURE
    CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT
                    RSNDMSK WSNDMSK ESNDMSK
                    RRETMSK WRETMSK ERETMSK
                    SELECB ERRNO RETCODE.
```

* ビット・マスクの長さは、次の式で決定されます。

```
((maximum socket number +32)/32 (drop the remainder))*4
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

MAXSOC

入力パラメーター。フルワード 2 進数フィールドであり、チェック対象のソケット記述子の最大値に 1 を加えた数値を指定します (TCP/IP for z/VSE は 0 から 8191 までの値のソケット記述子をサポートするためです)。

TIMEOUT

TIMEOUT が正の値の場合、その値は、選択が最大どれだけの間隔だけ待ってから完了するのかを示します。TIMEOUT-SECONDS が負の値の場合、SELECT 呼び出しは、ソケットが作動可能になるまでブロックします。ソケットをポーリングし、即時に戻るためには、TIMEOUT をゼロに設定します。

TIMEOUT は、次の 2 ワードの TIMEOUT で指定されます。

- TIMEOUT-SECONDS は、TIMEOUT フィールドの 1 番目のワードであり、タイムアウト値の秒の部分を表すコンポーネントです。
- TIMEOUT-MICROSEC は、TIMEOUT フィールドの 2 番目のワードであり、タイムアウト値のマイクロ秒の部分を表すコンポーネントです (0—999999)。

例えば、SELECTEX が 3.5 秒でタイムアウトになるようにするには、TIMEOUT-SECONDS を 3 に、TIMEOUT-MICROSEC を 500000 に設定します。

RSNDMSK

読み取り割り込みのチェックを制御するビット・マスク配列です。このパラメーターが指定されていないか、指定されたビット・マスクがゼロである場合、SELECT は読み取り割り込みをチェックしません。このビット・マスク配列の長さは、MAXSOC の値に依存します。

WSNDMSK

書き込み割り込みのチェックを制御するビット・マスク配列です。このパラメーターが指定されていないか、指定されたビット・マスクがゼロである場合、SELECT は書き込み割り込みをチェックしません。このビット・マスク配列の長さは、MAXSOC の値に依存します。

ESNDMSK
例外割り込みのチェックを制御するビット・マスク配列です。このパラメーターが指定されていないか、指定されたビット・マスクがゼロである場合、SELECT は例外割り込みをチェックしません。このビット・マスク配列の長さは、MAXSOC の値に依存します。

SELECB
ポストされると SELECTEX を完了させることになる ECB です。

ECB リストが指定される場合は、その ECB リストの最終項目の上位ビットを 1 に設定して、それが最終項目であることを示す必要があり、さらに、LIST キーワードを追加する必要があります。 ECB は、呼び出し元の 1 次アドレス・スペース内になければなりません。

注: リスト中に指定できる ECB の最大数は 254 です。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールドであり、RETCODE が負の場合、エラー番号を含みます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』を参照してください。

RETCODE
フルワード 2 進数フィールド

値	意味
>0	作動可能ソケットの数を示します。
0	SELECTEX の時間制限が満了した (ECB 値はゼロになります) か、呼び出し元の ECB の 1 つがポストされた (ECB 値は非ゼロになり、呼び出し元の記述子セットは 0 に設定されます) かのいずれかです。呼び出し元は、SELECTEX マクロを発行する前に、ECB 値をゼロに初期化しなければなりません。
-1	エラーです。ERRNO をチェックしてください。

RRETMSK
RSNDMSK が指定されている場合に SELECT が戻すビット・マスク配列です。このビット・マスク配列の長さは、MAXSOC の値に依存します。

WRETMSK
WSNDMSK が指定されている場合に SELECT が戻すビット・マスク配列です。このビット・マスク配列の長さは、MAXSOC の値に依存します。

ERETMSK
ESNDMSK が指定されている場合に SELECT が戻すビット・マスク配列です。このビット・マスク配列の長さは、MAXSOC の値に依存します。

SEND

SEND 呼び出しは、指定された接続済みソケット上でデータを送信します。

データグラム・ソケットの場合、SEND はデータグラム全体を、それが受信バッファーに収まる場合は、送信します。余分なデータは廃棄されます。

ストリーム・ソケットの場合、データは、データを分離する境界のない情報ストリームとして処理されます。例えば、プログラムが 1000 バイトを送信する必要がある場合、この関数のそれぞれの呼び出しは、1000 バイト全部までの任意のバイト数を送信することができ、送信されたバイト数が RETCODE に入れて戻されます。従って、ストリーム・ソケットを使用するプログラムでは、この呼び出しをループに入れ、全データが送信されるまでこの呼び出しを繰り返し発行する必要があります。

注: EBCDIC 入力データを ASCII に変換するサブルーチンについては、324 ページの『EZACIC04』を参照してください。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SEND'.
    01  S               PIC 9(4) BINARY.
    01  FLAGS           PIC 9(8) BINARY.
        88  NO-FLAG                VALUE IS 0.
        88  OOB                    VALUE IS 1.
        88  DONT-ROUTE             VALUE IS 4.
    01  NBYTE           PIC 9(8) BINARY.
    01  BUF             PIC X(length of buffer).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE
                    BUF ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、SEND を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S ハーフワード 2 進数であり、データを送信するソケットのソケット記述子を指定します。

FLAGS
フルワード 2 進数フィールドであり、0 に設定されなければなりません。

NBYTE
フルワード 2 進数であり、転送されるデータのバイト数に設定されます。

BUF 送信されるデータが入っているバッファーです。BUF のサイズは、NBYTE に指定されたサイズであるべきです。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
≥0	呼び出しは成功しました。値は、送信されたバイト数に設定されます。
-1	ERRNO のエラー・コードをチェックしてください。

SENDTO

SENDTO は、宛先アドレス・パラメーターを指定することを除いて SEND と同様です。

宛先アドレスを使用することで、ユーザーは、対象のソケットが接続されているかどうかにかかわらず、SENDTO 呼び出しにより、UDP ソケットへデータグラムを送信できます。

データグラム・ソケットの場合、SENDTO はデータグラム全体を、それが受信バッファーに収まる場合は、送信します。余分なデータは廃棄されます。

ストリーム・ソケットの場合、データは、データを分離する境界のない情報ストリームとして処理されます。例えば、プログラムが 1000 バイトを送信する必要がある場合、この関数のそれぞれの呼び出しは、1000 バイト全部までの任意のバイト数を送信することができ、送信されたバイト数が RETCODE に入れて戻されます。従って、ストリーム・ソケットを使用するプログラムでは、SENDTO をループに入れ、全データが送信されるまで呼び出しを繰り返す必要があります。

注: EBCDIC 入力データを ASCII に変換するサブルーチンについては、 324 ページの『EZACIC04』を参照してください。

重要: IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SENDTO'.
    01  S               PIC 9(4) BINARY.
    01  FLAGS.          PIC 9(8) BINARY.
        88  NO-FLAG         VALUE IS 0.
    01  NBYTE           PIC 9(8) BINARY.
    01  BUF             PIC X(length of buffer).
    01  NAME
        03  FAMILY      PIC 9(4) BINARY.
        03  PORT        PIC 9(4) BINARY.
        03  IP-ADDRESS  PIC 9(8) BINARY.
        03  RESERVED    PIC X(8).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE
                    BUF NAME ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、SENDTO を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S ハーフワード 2 進数であり、データを送信するソケットのソケット記述子に設定されます。

FLAGS
フルワードのフィールドであり、0 に設定されなければなりません。

NBYTE
フルワード 2 進数であり、送信するバイト数に設定されます。

BUF 送信されるデータが入っているバッファーを指定します。BUF のサイズは、NBYTE に指定されたサイズであるべきです。

NAME
入力パラメーター。IPv4 または IPv6 ターゲットのアドレス。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
説明

FAMILY
ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、ソケットにバインドされたポート番号を指定します。

IPv4-ADDRESS
フルワード 2 進数フィールドであり、ソケットの 32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

フィールド
説明

NAMELEN
IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる

値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、ソケットにバインドされたポート番号を指定します。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアント・マシンのソケットの 128 ビット IPv6 インターネット・アドレスが、ネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
≥0	呼び出しは成功しました。値は、送信されたバイト数に設定されます。
-1	ERRNO のエラー・コードをチェックしてください。

SETSOCKOPT

SETSOCKOPT 呼び出しは、ソケットに関連付けられたオプションを設定します。

SETSOCKOPT は、AF_INET または AF_INET6 ドメイン内のソケットに対してのみ呼び出すことができます。

OPTVAL パラメーターと OPTLEN パラメーターは、特定の設定コマンドで使用されるデータを渡すために使用されます。OPTVAL パラメーターは、設定コマンドが必要とするデータを含むバッファーを指します。OPTVAL パラメーターは、オプシ

ョンであり、コマンドがデータを必要としないときは、0 に設定することができます。OPTLEN パラメーターは、OPTVAL によって示されるデータのサイズに設定する必要があります。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SETSOCKOPT'.
    01  S               PIC 9(4) BINARY.
    01  OPTNAME         PIC 9(8) BINARY.
        88  SO-REUSEADDR  VALUE  4.
        88  SO-KEEPALIVE  VALUE  8.
        88  SO-LINGER     VALUE  128.
    01  OPTVAL          PIC 9(16) BINARY.
    01  OPTLEN          PIC 9(8) BINARY.
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S OPTNAME
                    OPTVAL OPTLEN ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'SETSOCKOPT' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S
ハーフワード 2 進数であり、オプションが設定されるソケットに設定されます。

OPTNAME
次のいずれかの値を指定します。

IP_ADD_MEMBERSHIP
アプリケーションが特定のインターフェース上のマルチキャスト・グループに参加できるようにするには、このオプションを使用します。このオプションでは、インターフェースを指定する必要があります。マルチキャスト・データグラムを受信するアプリケーションのみが、マルチキャスト・グループを結合する必要があります。これは IPv4 専用のソケット・オプションです。

IP_ADD_SOURCE_MEMBERSHIP
アプリケーションが特定のインターフェース上の特定のソース・アドレスを持つソース・マルチキャスト・グループに参加できるようにするには、このオプションを使用します。このオプションでは、インターフェースとソース・アドレスを指定する必要があります。マルチキャスト・データグラムを受け取るためには、アプリケーションがソース・マルチキャスト・グループに参加している必要があります。これは IPv4 専用のソケット・オプションです。

IP_BLOCK_SOURCE
指定された IPv4 ソース・アドレスと一致するソース・アドレスを持つマルチキャスト・パケットをアプリケーションでブロックできるようにするには、このオプションを使用します。このオプション

では、インターフェースとソース・アドレスを指定する必要があります。指定されたマルチキャスト・グループにあらかじめ参加しておく必要があります。これは IPv4 専用のソケット・オプションです。

IP_DROP_MEMBERSHIP
アプリケーションがマルチキャスト・グループまたはマルチキャスト・グループのすべてのソースから離脱できるようにするには、このオプションを使用します。これは IPv4 専用のソケット・オプションです。

IP_DROP_SOURCE_MEMBERSHIP
アプリケーションがソース・マルチキャスト・グループから離脱できるようにするには、このオプションを使用します。これは IPv4 専用のソケット・オプションです。

IP_MULTICAST_IF
このオプションは、ソケット・アプリケーションからアウトバウンド・マルチキャスト・データグラムを送信する際に使用する IPv4 インターフェース・アドレスを設定するために使用します。これは IPv4 専用のソケット・オプションです。

注**:** マルチキャスト・データグラムを送信できるインターフェースは、一度に 1 つのみです。

IP_MULTICAST_LOOP
送信ホスト自体が属するグループに送信されたマルチキャスト・データグラムについて、マルチキャスト・データグラムのコピーをループバックするかどうかを制御するには、このオプションを使用します。デフォルトでは、データグラムをループバックします。これは IPv4 専用のソケット・オプションです。

IP_MULTICAST_TTL
このオプションを使用して、発信マルチキャスト・データグラムの IP データグラム存続時間 (ホップ) を設定します。デフォルト値は '01'x です。これは、マルチキャストがローカル・サブネットでのみ使用可能であることを意味します。これは IPv4 専用のソケット・オプションです。

IP_UNBLOCK_SOURCE
このオプションは、所定の IPv4 マルチキャスト・グループに対して以前にブロックしたソースをアプリケーションでブロック解除できるようにするために使用します。このオプションでは、インターフェースとソース・アドレスを指定する必要があります。これは IPv4 専用のソケット・オプションです。

IPV6_JOIN_GROUP
このオプションを使用して、マルチキャスト・パケットの受信を制御し、ソケットがマルチキャスト・グループに参加することを指定します。これは IPv6 専用のソケット・オプションです。

IPV6_LEAVE_GROUP
このオプションを使用して、マルチキャスト・パケットの受信を制

御し、ソケットがマルチキャスト・グループから離脱することを指定します。これは IPv6 専用のソケット・オプションです。

IPV6_MULTICAST_HOPS
発信マルチキャスト・パケットに使用するホップ限界を設定するために使用します。これは IPv6 専用のソケット・オプションです。

IPV6_MULTICAST_IF
このオプションを使用して、ソケット・アプリケーションからアウトバウンド・マルチキャスト・データグラムを送信する際に使用する IPv6 インターフェースの索引を設定します。これは IPv6 専用のソケット・オプションです。

IPV6_MULTICAST_LOOP
送信ホスト自体が属するグループに対してデータグラムを送信する場合、ローカル配信のため、IP レイヤーによって発信インターフェース上でマルチキャスト・データグラムをループバックするかどうかを制御するには、このオプションを使用します。デフォルトでは、マルチキャスト・データグラムをループバックします。これは IPv6 専用のソケット・オプションです。

IPV6_UNICAST_HOPS
このオプションを使用して、発信ユニキャスト IPv6 パケットに使用するホップ限界を設定します。これは IPv6 専用のソケット・オプションです。

IPV6_V6ONLY
このオプションを使用して、ソケットを IPv6 パケットのみの送受信に制限するかどうかを設定します。デフォルトでは、IPv6 パケットのみの送受信に制限しません。これは IPv6 専用のソケット・オプションです。

MCAST_BLOCK_SOURCE
所定のソース・アドレスと一致するソース・アドレスを持つマルチキャスト・パケットをアプリケーションでブロックできるようにするには、このオプションを使用します。このオプションでは、インターフェース索引とソース・アドレスを指定する必要があります。指定されたマルチキャスト・グループにあらかじめ参加しておく必要があります。

MCAST_JOIN_GROUP
アプリケーションが特定のインターフェース上のマルチキャスト・グループに参加できるようにするには、このオプションを使用します。インターフェース索引を指定する必要があります。マルチキャスト・データグラムを受け取るには、アプリケーションがマルチキャスト・グループに参加する必要があります。

MCAST_JOIN_SOURCE_GROUP
アプリケーションが特定のインターフェースおよびソース・アドレスにあるソース・マルチキャスト・グループに参加できるようにするには、このオプションを使用します。インターフェース索引とソース・アドレスを指定する必要があります。特定のソース・アドレ

スからのみマルチキャスト・データグラムを受け取るには、アプリケーションがソース・マルチキャスト・グループに参加する必要があります。

MCAST_LEAVE_GROUP
アプリケーションがマルチキャスト・グループまたは所定のマルチキャスト・グループのすべてのソースから離脱できるようにするには、このオプションを使用します。

MCAST_LEAVE_SOURCE_GROUP
アプリケーションがソース・マルチキャスト・グループから離脱できるようにするには、このオプションを使用します。

MCAST_UNBLOCK_SOURCE
このオプションは、所定のマルチキャスト・グループに対して以前にブロックしたソースをアプリケーションでブロック解除できるようにするために使用します。このオプションでは、インターフェース索引とソース・アドレスを指定する必要があります。

SO-REUSEADDR
このオプションは、ソースの互換性のためにのみ用意されています。何もアクションは行いません。TCP/IP は、暗黙に、アドレスの即時再使用を許可しています。

SO-KEEPALIVE
このオプションは、ソースの互換性のためにのみ用意されています。何もアクションは行いません。この代わりに、ユーザーは一般的 TCP/IP 設定である SET PULSE_TIME=nnn を使用するべきです。

SO-LINGER
ソケットがクローズされるとき、送信できなかったデータを TCP/IP がどのように扱うのかを制御します。このオプションは、ストリーム・ソケットにのみ意味を持ちます。

- LINGER が使用可能に設定されていて、CLOSE が呼び出される場合、呼び出し側プログラムは、データが正常に送信されるか、接続がタイムアウトになるまでブロックされます。
- LINGER が使用不可に設定されている場合、CLOSE 呼び出しは呼び出し元をブロックせずに戻り、TCP/IP は、指定された期間の間、引き続きデータを送信します。通常、これはデータ送信を完了するのに十分な時間を提供しますが、LINGER オプションを使用すると、TCP/IP が待つのは OPTVAL LINGER に指定された時間だけなので、送信が正常に完了することは保証されません。

デフォルトでは使用不可に設定されます。

OPTVAL
OPTNAME に指定されたオプションをさらに定義するデータを含みます。

- OPTNAME が SO-REUSEADDR の場合、OPTVAL は 1 ワードの 2 進整数です。OPTVAL をゼロ以外の正の値に設定すると、オプションが使用可能に設定されます。OPTVAL をゼロに設定すると、オプションは使用不可になります。

- SO-LINGER の場合、OPTVAL は次の構造体を想定します。

```
ONOFF       PIC X(4).
LINGER      PIC 9(8) BINARY.
```

  ONOFF をゼロ以外の値に設定するとオプションが使用可能に設定され、ゼロに設定するとオプションは使用不可になります。LINGER 値は、TCP/IP が CLOSE 呼び出しの後もデータ送信を続行する時間を秒単位で設定します。

OPTLEN
フルワード 2 進数であり、OPTVAL に戻されるデータの長さを指定します。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

SHUTDOWN

ネットワーク接続を終了させる 1 つの方法は CLOSE 呼び出しを発行することです。この呼び出しは、接続を切断する前に、未処理のすべてのデータ伝送要求を完了させようとします。

HOW パラメーターが、シャットダウンするトラフィックの方向を指定します。

CLOSE 呼び出しが使用される場合、SETSOCKOPT OPTVAL LINGER パラメーターによって、システムが接続を解放する前に待機する時間が決定されます。例えば、LINGER 値が 30 秒の場合、システム・リソースは、CLOSE 呼び出しが発行された後も 30 秒間は、システム内にとどまります。大量の、トランザクション・ベースのシステムでは、これはパフォーマンスに大きな影響を与えることがあります。

SHUTDOWN 呼び出しを発行した場合、CLOSE 呼び出しを受け取ると、30 秒の遅延を待たずに接続をすぐにクローズできます。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SHUTDOWN'.
    01  S               PIC 9(4) BINARY.
    01  HOW             PIC 9(8) BINARY.
        88  END-BOTH      VALUE  2.
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.
```

```
PROCEDURE
    CALL 'EZASOKET' USING SOC-FUNCTION S HOW ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、SHUTDOWN を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S ハーフワード 2 進数であり、シャットダウンされるソケットのソケット記述子に設定されます。

HOW フルワード 2 進数フィールド。以下の値を設定できます。

値	説明
2 (END-BOTH)	それ以上の送信および受信操作を終了します。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

SOCKET

SOCKET 呼び出しは通信の端点を作成し、その端点を表すソケット記述子を戻します。

重要: IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'SOCKET'.
    01  AF              PIC 9(8) COMP VALUE 2.
    01  SOCTYPE         PIC 9(8) BINARY.
        88  STREAM        VALUE  1.
        88  DATAGRAM      VALUE  2.
    01  PROTO           PIC 9(8) BINARY.
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.
```

```
PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION AF SOCTYPE
                           PROTO ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、'SOCKET' を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

AF フルワード 2 進数フィールドであり、アドレス・ファミリーに設定されます。次のいずれかを指定します。

値 説明

'INET' または **10** 進数の **'2'**
変換されるアドレスが IPv4 アドレスであることを示します。

'INET6' または **10** 進数の **'19'**
変換されるアドレスが IPv6 アドレスであることを示します。

SOCTYPE
フルワード 2 進数フィールドであり、必要なソケットのタイプに設定されます。タイプは、以下のとおりです。

値 説明

1 ストリーム・ソケットは、信頼性があり、コネクション指向の、順次両方向バイト・ストリームを提供します。アウト・オブ・バンドのデータのメカニズムがサポートされます。

2 データグラム・ソケットは、信頼性が保証されていない、固定最大長のコネクションレス・メッセージであるデータグラムを提供します。データグラムでは、破壊、順序が狂った受信、紛失、または複数回の送達が起こる場合があります。

注**:** ロー・ソケットはサポートされません

PROTO
フルワード 2 進数フィールドであり、ソケットに使用されるプロトコルに設定されます。このフィールドが 0 に設定されている場合、デフォルトのプロトコルが使用されます。ストリームのデフォルトは TCP であり、データグラムのデフォルトは UDP です。このフィールドが 17 に設定されている場合、UDP プロトコルが使用されます。6 に設定されている場合、TCP プロトコルが使用されます。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値　　説明

> または = 0
　　新しいソケット記述子を含みます。

-1　　ERRNO のエラー・コードをチェックしてください。

TAKESOCKET

TAKESOCKET 呼び出しは他のプログラムからソケットを取得し、新規ソケットを作成します。

一般的には、子サーバーが、並行サーバーから取得したクライアント ID とソケット記述子を使用してこの呼び出しを発行します。GIVESOCKET および TAKESOCKET 呼び出しの使用についての説明は、271 ページの『GIVESOCKET』を参照してください。

注**:** TAKESOCKET が発行される場合、新規ソケット記述子が RETCODE で戻されます。S (ソケット記述子) パラメーターを必要とする、GETSOCKOPT などの、後続で発行する呼び出しでこの新規ソケット記述子を使用してください。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'TAKESOCKET'.
    01  SOCRECV         PIC 9(4) BINARY.
    01  CLIENT.
        03  DOMAIN      PIC 9(8) BINARY.
        03  NAME        PIC X(8).
        03  TASK        PIC X(8).
        03  RESERVED    PIC X(20).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION  SOCRECV CLIENT
                    ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
　　16 バイトの文字フィールドであり、TAKESOCKET を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

SOCRECV
　　ハーフワード 2 進数フィールドであり、取得されるソケットの記述子に設定されます。取得されるソケットは、並行サーバーによって渡されます。

CLIENT
　　ソケットを与えるプログラムのクライアント ID を指定します。CICS で

は、これらのパラメーターはリスナー・プログラムによって、
TAKESOCKET 呼び出しを発行するプログラムに渡されます。

- CICS では、情報の入手は EXEC CICS RETRIEVE を使用して行われます。

DOMAIN
入力パラメーター。フルワード 2 進数であり、ソケットを与えるプログラムのドメインに設定されます。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2、または AF_INET6 を示す 10 進数の 19 に設定されます。

注: TAKESOCKET は同じアドレス・ファミリーのソケットを、GIVESOCKET からのみ取得できます。

NAME
8 バイトの文字フィールドを指定し、ソケットを与えたプログラムの VSE パーティション ID に設定されます。

TASK
8 バイトの文字フィールドを指定し、ソケットを与えたタスクのタスク ID に設定されます。

RESERVED
20 バイトの予約フィールドです。このフィールドは必須であり、内部的にのみ使用されます。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
> または **=0**	新しいソケット記述子を含みます。
-1	ERRNO のエラー・コードをチェックしてください。

TERMAPI

この呼び出しは、INITAPI によって作成されたセッションを終了します。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'TERMAPI'.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、TERMAPI を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

WRITE

WRITE 呼び出しは、接続済みソケットでデータを書き込みます。この呼び出しは SEND に似ていますが、SEND で使用できる制御フラグがない点が異なります。

データグラム・ソケットの場合、WRITE 呼び出しはデータグラム全体を、それが受信バッファーに収まる場合は、書き込みます。

ストリーム・ソケットは、データを分離する境界のない情報ストリームのように動作します。例えば、プログラムで 1000 バイトを送信する必要がある場合、この関数へのそれぞれの呼び出しによって、任意のバイト数 (最大で 1000 バイト全体) を送信できます。送信されたバイト数が、RETCODE に入って戻されます。従って、ストリーム・ソケットを使用するプログラムでは、この呼び出しをループに入れ、全データが送信されるまでこの関数を繰り返し呼び出す必要があります。

EBCDIC 出力データを ASCII に変換するサブルーチンについては、324 ページの『EZACIC04』を参照してください。

COBOL の例

```
WORKING STORAGE
    01  SOC-FUNCTION    PIC X(16)  VALUE IS 'WRITE'.
    01  S               PIC 9(4) BINARY.
    01  NBYTE           PIC 9(8) BINARY.
    01  BUF             PIC X(length of buffer).
    01  ERRNO           PIC 9(8) BINARY.
    01  RETCODE         PIC S9(8) BINARY.

PROCEDURE
     CALL 'EZASOKET' USING SOC-FUNCTION S NBYTE BUF
                     ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、WRITE を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S　ハーフワード 2 進数フィールドであり、ソケット記述子に設定されます。

NBYTE
フルワード 2 進数フィールドであり、転送されるデータのバイト数に設定されます。

BUF　送信されるデータが入っているバッファーを指定します。

アプリケーションへ戻されるパラメーター値

ERRNO

フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
≥0	呼び出しは成功しました。正の戻りコードは、書き込まれたデータのバイト数を示します。
-1	ERRNO のエラー・コードをチェックしてください。

WRITEV

WRITEV 呼び出しでは、バッファー・セットからソケットにデータを書き込みます。

COBOL の例

```
   WORKING-STORAGE SECTION.
          01 SOC-FUNCTION        PIC X(16) VALUE 'WRITEV          '.
          01 S                   PIC 9(4) BINARY.
          01 ERRNO               PIC 9(8) BINARY.
          01 RETCODE             PIC S9(8) BINARY.
          01 IOVCNT              PIC 9(8) BINARY.
          01 IOV.
             03 BUFFER-ENTRY OCCURS 5 TIMES.
                05 IOV-POINTER       USAGE IS POINTER.
                05 RESERVED          PIC X(4).
                05 IOV-LENGTH        PIC 9(8) BINARY.
          01 HEAPID              PIC S9(9) BINARY VALUE IS 0.

       PROCEDURE DIVISION.


          MOVE 50 TO IOV-LENGTH(1).
          MOVE 50 TO IOV-LENGTH(2).
          MOVE 50 TO IOV-LENGTH(3).
          MOVE 50 TO IOV-LENGTH(4).
          MOVE 50 TO IOV-LENGTH(5).
          CALL 'CEEGTST' USING HEAPID, IOV-LENGTH(1), IOV-POINTER(1),
                               FC.
          CALL 'CEEGTST' USING HEAPID, IOV-LENGTH(2), IOV-POINTER(2),
                               FC.
          CALL 'CEEGTST' USING HEAPID, IOV-LENGTH(3), IOV-POINTER(3),
                               FC.
          CALL 'CEEGTST' USING HEAPID, IOV-LENGTH(4), IOV-POINTER(4),
                               FC.
          CALL 'CEEGTST' USING HEAPID, IOV-LENGTH(5), IOV-POINTER(5),
                               FC.

        * Call subroutine to fill the IOV structure
            CALL 'subroutine' USING IOV.

            CALL 'EZASOKET' USING SOC-FUNCTION S IOV IOVCNT
                         ERRNO RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

SOC-FUNCTION
16 バイトの文字フィールドであり、WRITEV を含みます。このフィールドは、左寄せにして右にブランクが埋め込まれます。

S ハーフワードの 2 進数であり、データを書き込むソケットのソケット記述子に設定されます。

IOV IOVCNT の値と等しい数の構造体の 3 つのフルワード構造の配列。

構造体の形式は、次のとおりです。

- フルワード 1: データ・バッファーのアドレス。
- フルワード 2: 予約済み
- フルワード 3: フルワード 1 で参照されるデータ・バッファーの長さ

IOVCNT
フルワードの 2 進数フィールドであり、この呼び出しに提供されるデータ・バッファー数を指定します。最大は 120 です。

アプリケーションへ戻されるパラメーター値

ERRNO
フルワード 2 進数フィールド。RETCODE が負の場合、このフィールドにはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
≥0	書き込まれたバイト数。
-1	エラーが発生しました。ERRNO のエラー・コードをチェックしてください。

ソケット呼び出しインターフェース用のデータ変換プログラムの使用

ソケット呼び出しに加えて、データ変換のために次のユーティリティー・プログラムを使用できます。

データ変換

TCP/IP ホストおよびネットワークは ASCII データ表記を使用します。 TCP/IP for z/VSE およびそのサブシステムは EBCDIC データ表記を使用します。一方の表記から他方の表記にデータを変換しなければならない状況では、以下のユーティリティー・プログラムを使用できます。

- EZACIC04 — EBCDIC データを ASCII データに変換します。
- EZACIC05 — ASCII データを EBCDIC データに変換します。

ビット・ストリング処理

C 言語では、フラグ、スイッチ設定などにビット・ストリングが頻繁に使用されます。TCP/IP はビット・ストリングを頻繁に使用します。しかし、ビット・ストリングを COBOL でデコードするのは難しいので、TCP/IP には以下が組み込まれています。

- EZACIC06 — ビット・マスクを文字配列に、文字配列をビット・マスクに変換します。
- EZACIC08 — GETHOSTBYNAME または GETHOSTBYADDR で戻される HOSTENT 構造体内の可変長のアドレス・リストを解釈します。
- EZACIC09 — GETADDRINFO により返される ADDRINFO 構造を解釈します。

EZACIC04

EZACIC04 プログラムは、EBCDIC データを ASCII データに変換するために使用されます。

COBOL の例

```
WORKING STORAGE
    01  OUT-BUFFER   PIC X(length of output).
    01  LENGTH       PIC 9(8) BINARY.

PROCEDURE
     CALL 'EZACIC04' USING OUT-BUFFER LENGTH.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

OUT-BUFFER
以下のデータを収容するバッファーです。

- 呼び出し時 – EBCDIC データ
- 戻り時 - ASCII データ

LENGTH
変換されるデータの長さを指定します。

EZACIC05

EZACIC05 プログラムは、ASCII データを EBCDIC データに変換するために使用されます。EBCDIC データは、COBOL、PL/I、およびアセンブラー言語のプログラムで必要とされます。

COBOL の例

```
WORKING STORAGE
    01  IN-BUFFER    PIC X(length of output)
    01  LENGTH       PIC 9(8) BINARY VALUE

PROCEDURE
     CALL 'EZACIC05' USING IN-BUFFER LENGTH.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

IN-BUFFER
以下のデータを収容するバッファーです。

- 呼び出し時 – ASCII データ
- 戻り時 - EBCDIC データ

LENGTH
変換されるデータの長さを指定します。

EZACIC06

SELECT 呼び出しは、ビット・ストリングを使用して、テストするソケットを指定し、テスト結果を戻します。ビット・ストリングを COBOL で処理するのは難しいので、SELECT 呼び出しで使用するために、アセンブラー言語プログラム EZACIC06 を使用してビット・ストリングを文字ストリングに変換することができます。

COBOL の例

```
WORKING STORAGE
    01  CHAR-MASK.
        05 CHAR-STRING              PIC X(nn).

    01  CHAR-ARRAY  REDEFINES CHAR-MASK.
           05  CHAR-ENTRY-TABLE  OCCURS nn TIMES.
               10  CHAR-ENTRY       PIC X(1).
    01  BIT-MASK.
        05 BIT-ARRAY-FWDS           PIC 9(16) COMP.

    01  BIT-FUNCTION-CODES.
        05  CTOB                    PIC X(4) VALUE 'CTOB'.
        05  BTOC                    PIC X(4) VALUE 'BTOC'.

    01  BIT-MASK-LENGTH             PIC 9(8) COMP VALUE 50.
    01  CHAR-STRING-LENGTH          PIC 9(8) COMP VALUE 64.



PROCEDURE CALL (to convert from character to binary)
     CALL 'EZACIC06' USING CTOB
                           BIT-MASK
                           CHAR-MASK
                           CHAR-STRING-LENGTH
                           RETCODE.


PROCEDURE CALL (to convert from binary to character)
     CALL 'EZACIC06' USING BTOC
                           BIT-MASK
                           CHAR-MASK
                           BIT-MASK-LENGTH
                           RETCODE.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

TOKEN
16 文字の ID です。この ID は必須であり、リスト中の最初のパラメーターでなければなりません。

CHAR-MASK
文字配列を指定します。ここで、*nn* は、配列内のソケットの最大数です。

BIT-MASK
SELECT 呼び出し用に変換されるビット・ストリングを指定します。ビットは、右から左に順序付けられ、右端のビットがソケット 0 を表します。文字配列内でのソケット位置は、1 から順に索引付けられ、ソケット 0 が、文字配列中の索引番号 1 になります。このことを念頭に、文字位置をオンまたはオフにしてください。

COMMAND
BTOC—ビット・ストリングから文字配列への変換を指定します。

CTOB—文字配列からビット・ストリングへの変換を指定します。

BIT-MASK-LENGTH
ビット・マスクの長さを指定します。

CHAR-STRING-LENGTH
文字マスクの長さを指定します。

アプリケーションへ戻されるパラメーター値

RETCODE
2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

SELECT 呼び出しを使用して、ソケット 0、5、および 9 をテストする場合で、かつ、文字配列を使用してソケットを表す場合は、文字配列内の該当する文字を 1 に設定する必要があります。この例では、文字配列内の索引 1、6、および 10 の位置が 1 に設定されます。その後、COMMAND パラメーターを CTOB に設定して EZACIC06 を呼び出します。EZACIC06 が戻ると、ビット 0、5、および 9 (番号は右から順に付与されます) がオンに設定されたフルワードが BIT-MASK に戻され、これを SELECT 呼び出しで使用できます。これらの命令は、以下の例に示すようにビット・ストリングを処理します。

```
MOVE ZEROS TO CHAR-STRING.
MOVE '1'TO CHAR-ENTRY(1), CHAR-ENTRY(6), CHAR-ENTRY(10).
CALL 'EZACIC06' USING CTOB BIT-MASK CHAR-MASK
      CHAR-STRING-LENGTH RETCODE.
MOVE BIT-MASK TO ....
```

SELECT 呼び出しが戻った際に、ソケット・アクティビティーのビット・マスク・ストリングをチェックする場合は、以下の命令を入力します。

```
MOVE ..... TO BIT-MASK.
CALL 'EZACIC06' USING BTOC BIT-MASK CHAR-MASK
         BIT-MASK-LENGTH RETCODE.
PERFORM TEST-SOCKET THRU TEST-SOCKET-EXIT  VARYING IDX
```

```
     FROM 1 BY 1 UNTIL IDX EQUAL 10.

TEST-SOCKET.
      IF CHAR-ENTRY(IDX) EQUAL '1'
           THEN PERFORM SOCKET-RESPONSE THRU SOCKET-RESPONSE-EXIT
           ELSE NEXT SENTENCE.
TEST-SOCKET-EXIT.
      EXIT.
```

EZACIC08

GETHOSTBYNAME および GETHOSTBYADDR 呼び出しは、構造体 HOSTENT を戻す C ソケット呼び出しから派生したものです。1 つの TCP/IP ホストは、複数の別名およびホスト IP アドレスを持つことができます。

TCP/IP は、間接アドレッシングを使用して、GETHOSTBYADDR および GETHOSTBYNAME 呼び出しから戻される HOSTENT 構造体内の可変の個数の別名および IP アドレスを接続します。

HOSTENT 構造体は、PL/I またはアセンブラー言語を使用してコーディングする場合は、比較的簡単に処理できます。しかし、COBOL でコーディングする場合、HOSTENT を扱うのは難しく、それを処理するために EZACIC08 サブルーチンを使用するべきです。

これは、次のように機能します。

- GETHOSTBYADDR または GETHOSTBYNAME は、HOSTENT 構造体を戻します。この構造体は、別名と IP アドレスのリストを間接的にアドレス指定します。
- GETHOSTBYADDR または GETHOSTBYNAME が戻ったら、プログラムは EZACIC08 を呼び出し、HOSTENT 構造体のアドレスを渡します。EZACIC08 は、この構造体を処理し、次のものを戻します。
  1. ホスト名の長さ (ある場合)
  2. ホスト名
  3. ホストの別名の数
  4. 別名シーケンス番号
  5. 別名の長さ
  6. 別名
  7. ホスト IP アドレスのタイプ (AF_INET に対しては常に 2)
  8. ホスト IP アドレスの長さ (AF_INET に対しては常に 4)
  9. このホストのホスト IP アドレスの数
  10. ホスト IP アドレスのシーケンス番号
  11. ホスト IP アドレス
- GETHOSTBYADDR または GETHOSTBYNAME 呼び出しが、複数の別名またはホスト IP アドレスを戻した場合 (上記のステップ 3 および 9)、アプリケーション・プログラムは、すべての別名およびホスト IP アドレスが取得されるまで、EZACIC08 の呼び出しを繰り返す必要があります。

COBOL の例

```
WORKING STORAGE

    01  HOSTENT-ADDR       PIC 9(8) BINARY.
    01  HOSTNAME-LENGTH    PIC 9(4) BINARY.
    01  HOSTNAME-VALUE     PIC X(255)
    01  HOSTALIAS-COUNT    PIC 9(4) BINARY.
    01  HOSTALIAS-SEQ      PIC 9(4) BINARY.
    01  HOSTALIAS-LENGTH   PIC 9(4) BINARY.
    01  HOSTALIAS-VALUE    PIC X(255)
    01  HOSTADDR-TYPE      PIC 9(4) BINARY.
    01  HOSTADDR-LENGTH    PIC 9(4) BINARY.
    01  HOSTADDR-COUNT     PIC 9(4) BINARY.
    01  HOSTADDR-SEQ       PIC 9(4) BINARY.
    01  HOSTADDR-VALUE     PIC 9(8) BINARY.
    01  RETURN-CODE        PIC 9(8) BINARY.

PROCEDURE

   CALL 'EZASOKET' USING 'GETHOSTBYxxxx'
                   HOSTENT-ADDR
                   RETCODE.

   ここで、xxxx は ADDR または NAME です。

   CALL 'EZACIC08' USING HOSTENT-ADDR HOSTNAME-LENGTH
                   HOSTNAME-VALUE HOSTALIAS-COUNT HOSTALIAS-SEQ
                   HOSTALIAS-LENGTH HOSTALIAS-VALUE
                   HOSTADDR-TYPE HOSTADDR-LENGTH HOSTADDR-COUNT
                   HOSTADDR-SEQ HOSTADDR-VALUE RETURN-CODE
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

HOSTENT-ADDR

フルワード 2 進数フィールドであり、HOSTENT 構造体のアドレスを含んでいなければなりません (GETHOSTBY*xxxx* 呼び出しによって戻されたとおりに)。この変数は、GETHOSTBYADDR および GETHOSTBYNAME ソケット呼び出しの HOSTENT 変数と同じです。

HOSTALIAS-SEQ

ハーフワードのフィールドであり、別名リストに索引を付けるために EZACIC08 によって使用されます。EZACIC08 は、呼び出されると、現在の HOSTALIAS-SEQ 値に 1 を加算し、結果の値を別名テーブルの索引として使用します。従って、任意の GETHOSTBYxxxx について、このフィールドは最初の EZACIC08 呼び出しでは 0 に設定されなければなりません。後続のすべての EZACIC08 呼び出しでは、このフィールドには、直前の呼び出しから戻された HOSTALIAS-SEQ の数値を入れるようにするべきです。

HOSTADDR-SEQ

ハーフワードのフィールドであり、IP アドレスのリストに索引を付けるために EZACIC08 によって使用されます。EZACIC08 は、呼び出されると、現在の HOSTADDR-SEQ 値に 1 を加算し、結果の値を IP アドレスのテーブルの索引として使用します。従って、任意の GETHOSTBYxxxx について、このフィールドは最初の EZACIC08 呼び出しでは 0 に設定されなけ

ればなりません。後続のすべての EZACIC08 呼び出しでは、このフィールドには、直前の呼び出しから戻された HOSTADDR-SEQ の数値を入れるようにするべきです。

アプリケーションへ戻されるパラメーター値

HOSTNAME-LENGTH
ハーフワード 2 進数フィールドであり、ホスト名 (ホスト名が戻された場合) の長さを含みます。

HOSTNAME-VALUE
255 バイトの文字ストリングであり、ホスト名 (ホスト名が戻された場合) を含みます。

HOSTALIAS-COUNT
ハーフワード 2 進数フィールドであり、戻された別名の数を含みます。

HOSTALIAS-SEQ
ハーフワード 2 進数フィールドであり、現在 HOSTALIAS-VALUE に入っている別名のシーケンス番号を含みます。

HOSTALIAS-LENGTH
ハーフワード 2 進数フィールドであり、現在 HOSTALIAS-VALUE に入っている別名の長さを含みます。

HOSTALIAS-VALUE
255 バイトの文字ストリングであり、呼び出しのこのインスタンスから戻される別名を含みます。別名の長さは HOSTALIAS-LENGTH に入っています。

HOSTADDR-TYPE
ハーフワード 2 進数フィールドであり、ホスト・アドレスのタイプを含みます。FAMILY タイプ AF_INET に対して、HOSTADDR-TYPE は常に 2 です。

HOSTADDR-LENGTH
ハーフワード 2 進数フィールドであり、現在 HOSTADDR-VALUE に入っているホスト IP アドレスの長さを含みます。FAMILY タイプ AF_INET に対して、HOSTADDR-LENGTH は常に 4 に設定されます。

HOSTADDR-COUNT
ハーフワード 2 進数フィールドであり、呼び出しのこのインスタンスから戻されるホスト IP アドレスの数を含みます。

HOSTADDR-SEQ
ハーフワード 2 進数フィールドであり、現在 HOSTADDR-VALUE に入っているホスト IP アドレスのシーケンス番号を含みます。

HOSTADDR-VALUE
フルワード 2 進数フィールドであり、ホスト IP アドレスを含みます。

RETURN-CODE
フルワード 2 進数フィールドであり、EZACIC08 戻りコードを含みます。

値	説明
0	正常終了しました。

-1 HOSTENT アドレスが無効です。

EZACIC09

GETADDRINFO 呼び出しは、RES という構造体を返す C ソケット呼び出しから派生したものです。1 つの TCP/IP ホストは、複数セットの NAMES を持つことができます。TCP/IP は、間接アドレッシングを使用して、GETADDRINFO 呼び出しから返される RES 構造体内の可変の個数の NAMES に接続します。RES 構造体は、PL/I またはアセンブラー言語を使用してコーディングする場合は、比較的簡単に処理できます。しかし、COBOL でコーディングする場合、RES を扱うのは難しく、それを処理するために EZACIC09 サブルーチンを使用するべきです。

これは、次のように機能します。

1. GETADDRINFO は、ソケット・アドレス構造体のリストを間接的にアドレス指定する RES 構造体を返します。
2. GETADDRINFO から RES 構造が戻されると、NEXT 引数で参照されるときに、プログラムは EZACIC09 を呼び出し、次のアドレス情報構造体のアドレスを渡します。EZACIC09 は、この構造体を処理し、次のものを戻します。
   a. ソケット・アドレス構造体
   b. 次のアドレス情報構造体
3. GETADDRINFO 呼び出しが複数のソケット・アドレス構造体を返す場合、アプリケーション・プログラムは、すべてのソケット・アドレス構造体が取得されるまで、EZACIC09 への呼び出しを繰り返すことになります。

COBOL の例

```
WORKING-STORAGE SECTION.
*
* Variables used for the GETADDRINFO call
*
 01 getaddrinfo-parms.
    02 node-name pic x(255).
    02 node-name-len pic 9(8) binary.
    02 service-name pic x(32).
    02 service-name-len pic 9(8) binary.
    02 canonical-name-len pic 9(8) binary.
    02 ai-passive pic 9(8) binary value 1.
    02 ai-canonnameok pic 9(8) binary value 2.
    02 ai-numerichost pic 9(8) binary value 4.
    02 ai-numericserv pic 9(8) binary value 8.
    02 ai-v4mapped pic 9(8) binary value 16.
    02 ai-all pic 9(8) binary value 32.
    02 ai-addrconfig pic 9(8) binary value 64.
*
* Variables used for the EZACIC09 call
*
 01 ezacic09-parms.
    02 res usage is pointer.
    02 res-name-len pic 9(8) binary.
    02 res-canonical-name pic x(256).
    02 res-name usage is pointer.
    02 res-next-addrinfo usage is pointer.
*
* Socket address structure
*
 01 server-socket-address.
    05 server-family pic 9(4) Binary Value 19.
```

```
    05 server-port pic 9(4) Binary Value 9997.
    05 server-flowinfo pic 9(8) Binary Value 0.
    05 server-ipaddr.
       10 filler pic 9(16) binary value 0.
       10 filler pic 9(16) binary value 0.
    05 server-scopeid pic 9(8) Binary Value 0.

LINKAGE SECTION.
  01 L1.
     03 HINTS-ADDRINFO.
     05 HINTS-AI-FLAGS PIC 9(8) BINARY.
     05 HINTS-AI-FAMILY PIC 9(8) BINARY.
     05 HINTS-AI-SOCKTYPE PIC 9(8) BINARY.
     05 HINTS-AI-PROTOCOL PIC 9(8) BINARY.
     05 FILLER PIC 9(8) BINARY.
     05 FILLER PIC 9(8) BINARY.
     05 FILLER PIC 9(8) BINARY.
     05 FILLER PIC 9(8) BINARY.
  03 HINTS-ADDRINFO-PTR USAGE IS POINTER.
  03 RES-ADDRINFO-PTR USAGE IS POINTER.
*
* RESULTS ADDRESS INFO
*
01 RESULTS-ADDRINFO.
   05 RESULTS-AI-FLAGS PIC 9(8) BINARY.
   05 RESULTS-AI-FAMILY PIC 9(8) BINARY.
   05 RESULTS-AI-SOCKTYPE PIC 9(8) BINARY.
   05 RESULTS-AI-PROTOCOL PIC 9(8) BINARY.
   05 RESULTS-AI-ADDR-LEN PIC 9(8) BINARY.
   05 RESULTS-AI-CANONICAL-NAME USAGE IS POINTER.
   05 RESULTS-AI-ADDR-PTR USAGE IS POINTER.
   05 RESULTS-AI-NEXT-PTR USAGE IS POINTER.
*
* SOCKET ADDRESS STRUCTURE FROM EZACIC09.
*
01 OUTPUT-NAME-PTR USAGE IS POINTER.
01 OUTPUT-IP-NAME.
   03 OUTPUT-IP-FAMILY PIC 9(4) BINARY.
   03 OUTPUT-IP-PORT PIC 9(4) BINARY.
   03 OUTPUT-IP-SOCK-DATA PIC X(24).
   03 OUTPUT-IPV4-SOCK-DATA REDEFINES OUTPUT-IP-SOCK-DATA.
      05 OUTPUT-IPV4-IPADDR PIC 9(8) BINARY.
      05 FILLER PIC X(20).
   03 OUTPUT-IPV6-SOCK-DATA REDEFINES OUTPUT-IP-SOCK-DATA.
      05 OUTPUT-IPV6-FLOWINFO PIC 9(8) BINARY.
      05 OUTPUT-IPV6-IPADDR.
         10 FILLER PIC 9(16) BINARY.
         10 FILLER PIC 9(16) BINARY.
      05 OUTPUT-IPV6-SCOPEID PIC 9(8) BINARY.

PROCEDURE DIVISION USING L1.
*
* Get and address from the resolver.
*
     move 'yournodename' to node-name.
     move 12 to node-name-len.
     move spaces to service-name.
     move 0 to service-name-len.
     move af-inet6 to hints-ai-family.
     move 49 to hints-ai-flags move 0 to hints-ai-socktype.
     move 0 to hints-ai-protocol.
     set address of results-addrinfo to res-addrinfo-ptr.
     set hints-addrinfo-ptr to address of hints-addrinfo.
     call 'EZASOKET' using soket-getaddrinfo node-name node-name-len
mm                     service-name service-name-len hints-addrinfo-ptr
                     res-addrinfo-ptr canonical-name-len errno retcode.
*
* Use EZACIC09 to extract the IP address
```

```
*
     set address of results-addrinfo to res-addrinfo-ptr.
     set res to address of results-addrinfo.
     move zeros to res-name-len.
     move spaces to res-canonical-name.
     set res-name to nulls.
     set res-next-addrinfo to nulls.
     call 'EZACIC09' using res res-name-len res-canonical-name
                    res-name res-next-addrinfo retcode.
     set address of output-ip-name to res-name.
     move output-ipv6-ipaddr to server-ipaddr.
```

対応する PL/I およびアセンブラー言語の宣言については、 231 ページの『パラメーター記述の変換』 を参照してください。

アプリケーションが設定するパラメーター値

RES フルワード 2 進数フィールドであり、ADDRINFO 構造体のアドレスを含んでいなければなりません (GETADDRINFO 呼び出しによって戻されたとおりに)。この変数は、GETADDRINFO ソケット呼び出しの RES 変数と同じです。

RES-NAME-LEN
GETADDRINFO 呼び出しにより返されるソケット・アドレス構造体の長さが入る、フルワード 2 進フィールド。

アプリケーションへ戻されるパラメーター値

RES-CANONICAL-NAME
正規名を十分に保持できる大きさがあるフィールド。フィールドの最大サイズは 256 バイトです。正規名長さフィールドは、GETADDRINFO 呼び出しにより返される正規名の長さを示します。

RES-NAME
後続のソケット・アドレス構造体のアドレス。

RES-NEXT
次のアドレス情報構造体のアドレス。RETCODE

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、EZACIC09 戻りコードを含みます。

値	説明
≥0	呼び出しは成功しました。
-1	RES アドレスが無効です。

第 13 章 マクロ・アプリケーション・プログラミング・インターフェース (EZASMI API) の使用

このトピックでは、System/390 アセンブラー言語で作成される TCP/IP アプリケーション・プログラム用のマクロ API について説明します。

このマクロ・インターフェースは、再入可能モジュールを生成するために使用できます。

本章には、以下のトピックがあります。

- 環境に関する制約事項およびプログラミング要件
- API マクロ用ストレージの定義
- 共通パラメーターについて
- ストリーム・ソケットの特性
- タスク管理および非同期の関数処理
- 非請求イベント出口ルーチンの使用
- エラー・メッセージおよび戻りコード
- アセンブラー・プログラム用のマクロ

環境に関する制約事項およびプログラミング要件

マクロ・ソケット API には、以下の制約事項が適用されます。

- EZASMI API は、ICCF 疑似パーティションで実行しているプログラムで使用することはできません。
- ロック

  これらの呼び出しを発行するときには、ロックが保持されているべきではありません。
- INITAPI/TERMAPI マクロ

  INITAPI/TERMAPI マクロは、同じタスク下で発行されなければなりません。
- ストレージ

  ソケット呼び出しから戻されるデータを収容する目的のために獲得するストレージは、ソケット呼び出し時のアプリケーション・プログラム状況ワード (PSW) と同じキーで取得されなければなりません。
- CICS がストレージ保護で動作しているときに CICS トランザクション内で EZASMI マクロ API を使用する場合、このマクロ API を使用するすべてのプログラムは、EXECKEY(CICS) で定義されている必要があります。それらのプログラムにリンクされているプログラムについても同様です。トランザクション定義の TASKDATAKEY(CICS) は不要です。
- アドレス指定モード (AMODE) の考慮事項

EZASMI マクロ API は、呼び出し元が 31 ビット AMODE である間に起動されなければなりません。

- CICS トランザクション内でマクロ API を使用する場合、EZA 「タスク関連ユーザー出口」(TRUE) が活動化された後でないと、これらのトランザクションは実行できません。この TRUE を活動化する方法について詳しくは、 93 ページの『EZA インターフェースに関する CICS の考慮事項』を参照してください。
- アプリケーションで使用する TCP/IP および SSL の実装を選択する方法については、 95 ページの『第 10 章 使用する TCP/IP と SSL の実装を選択』を参照してください。

EZASMI マクロ・アプリケーション・プログラム・インターフェース (API)

このセクションでは、High Level Assembler 言語で作成される TCP/IP アプリケーション・プログラム用の EZASMI マクロ API について説明します。各ソケット呼び出しごとに、フォーマットとパラメーターを説明します。

注:

1. このインターフェースでは再入可能コードがサポートされます。
2. レジスター規則: レジスター 0、1、14、15 は、インターフェースによって使用されるので、必要であれば起動前に保管が必要です。レジスター 13 は、呼び出し元が用意する 72 バイトの保管域を示さなければなりません。

API マクロ用ストレージの定義

マクロ API は、タスク・ストレージ域を必要とします。

タスク・ストレージ域は、指定された接続を介して通信するすべてのソケット・ユーザーに認識されていて、それらのソケット・ユーザーからアドレッシング可能でなければなりません。アプリケーションと TCP/IP の間には 1 つの接続が稼働します。ストレージを編成する方法で最も一般的であるのは、各 VSE サブタスクごとに 1 つの接続を割り当てる方法です。1 つのタスクまたは接続内に、ソケットを使用する複数のモジュールがある場合、タスク・ストレージのアドレスを各ユーザーに提供する必要があります。

タスク・ストレージのアドレスを定義する方法として、以下の 2 つの方法があります。

- プログラム・コードの一部として、STORAGE=CSECT にして EZASMI TYPE=TASK 命令をコーディングする。こうすると、プログラムは再入不可になりますが、コードはシンプルになります。
- プログラム・コードの一部として、STORAGE=DSECT にして EZASMI TYPE=TASK 命令をコーディングする。この命令の展開で、ストレージ域の長さと等価のフィールド TIELENTH が生成されます。これを使用して VSE GETVIS 要求を発行することで、必要なストレージを割り振ります。使用する前にこのストレージ域がバイナリー 0 にクリアされるようにしてください。

定義するプログラムは、このストレージのアドレスを、この接続を使用する他のすべてのプログラムに対して使用可能にする必要があります。これらのタスクを実行

するプログラムは、STORAGE=DSECT にした EZASMI TYPE=TASK でストレージ・マッピングを定義する必要があります。

EZASMI TYPE=TASK マクロは、1 つの接続について 1 つのみのパラメーター・リストを生成します。プログラムは、以下のフォーマットを使用して、各関数呼び出し用に固有のパラメーター・リスト・ストレージ域を構築できます。

```
BINDPRML   EZASMI    MF=L       This will generate the storage used for
                                building the parm list in the following BIND call
           EZASMI    TYPE=BIND,                                        *
                     S=SOCKDESC,                                       *
                     NAME=NAMEID,                                      *
                     ERRNO=ERRNO,                                      *
                     RETCODE=RETCODE,                                  *
                     ECB=ECB1,                                         *
                     MF=(E,BINDPRML)
```

これは、非同期 BIND マクロの例であり、MF=L マクロを使用してパラメーター・リストを生成します。すべてのマクロ呼び出しに共通するフィールド (例えば、RETCODE および ERRNO) は、それぞれの未解決呼び出しで固有でなければなりません。

TCP/IP への複数の接続を単一のタスクから作成できます。それらの接続のそれぞれが、他の接続から独立して機能し、独自のタスク・インターフェース・エレメント (TIE) によって識別されます。TASK パラメーターを使用すると、明示的に TIE を参照できます。TASK パラメーターが組み込まれていない場合、マクロは EZASMI TYPE=TASK マクロで生成される TIE を使用します。

```
TIE1   DS XL(TIELENTH)         Length of TIE

EZASMI   TYPE=INITAPI,                                          *
         MAXSOC=MAX75,                                          *
         ERRNO=ERRNO,                                           *
         RETCODE=RETCODE,                                       *
         APITYPE=2,                                             *
         MAXSNO=MAXS,                                           *
         TASK=TIE1                                              *

EZASMI   TYPE=SOCKET,                                           *
         AF='INET',                                             *
         SOCTYPE='STREAM',                                      *
         ERRNO=ERRNO,                                           *
         RETCODE=RETCODE,
         TASK=TIE1
```

この例では、EZASMI TYPE=TASK マクロによって生成される TIE ではなく、TIE TIE1 が接続用に使用されます。

共通パラメーターについて

このセクションでは、ここに記載される各マクロに共通するパラメーターと概念について説明します。

パラメーター
: 説明

address
: パラメーターの値が入ったフィールドの名前です。次の例は、SOCKNO が 2 に設定された BIND マクロを示します。

```
MVC   SOCKNO,=H'2'
EZASMI TYPE=BIND,S=SOCKNO
```

**indaddr*

パラメーターを含むフィールドのアドレスが入ったアドレス・フィールドの名前です。次の例は、上の例と同じ結果になります。

```
MVC   SOCKNO,=H'2'
LA    0,SOCKNO
ST    0,SOCKADD
EZASMI TYPE=BIND,S=*SOCKADD
```

(reg) 汎用レジスターの名前 (番号と等価) または番号です。レジスター 0、1、14、15 を使用しないでください。次の例は、上の 2 つの例と同じ結果になります。

```
MVC   SOCKNO,=H'2'
LA    3,SOCKNO
EZASMI TYPE=BIND,SOCKNO=(3)
```

'value'

パラメーターのリテラル値。例えば、AF='INET' です。

ストリーム・ソケットの特性

ストリーム・ソケットの場合、データは、データを分離する境界のない情報ストリームとして処理されます。例えば、アプリケーション A および B がストリーム・ソケットで接続され、アプリケーション A が 1000 バイトを送信する場合、SEND 関数のそれぞれの呼び出しは、1 バイト、または 10 バイト、あるいは 1000 バイト全体を戻すことができ、送信されたバイト数は RETCODE に戻されます。従って、ストリーム・ソケットを使用するプログラムでは、READ 呼び出しまたは SEND 呼び出しをループに入れ、全データが送信または受信されるまで繰り返す必要があります。

タスク管理および非同期の関数処理

EZASMI ソケット・インターフェースでは非同期操作が許可されますが、デフォルトでは、マクロ要求を発行するタスクは、要求された関数が完了するまで WAIT 状態に置かれます。完了時点で、発行元のタスクは再開し、実行を続けます。

要求が処理される間に発行元のタスクが WAIT 状態に置かれないようにするには、非同期の関数処理を使用します。

どのように機能するか

マクロ API は、2 つの形式で非同期の関数処理を提供しています。どちらの形式を使用しても、関数要求が TCP/IP に送信されたらすぐに、システムはアプリケーションに制御を戻します。2 つの形式は、関数が完了したときにそれがアプリケーションにどのように通知されるのかという点で異なります。

ECB 方式

各ソケット呼び出しごとに VSE イベント制御ブロック (ECB) を渡すことができます。ソケット呼び出しは、即時にプログラムに制御を戻し、呼び出しが完了したときに ECB をポストします。

出口方式

INITAPI 呼び出しを使用する出口ルーチンの入り口点を指定できます。個々のソケット呼び出しはプログラムに即時に制御を戻し、ソケット呼び出しは、そのソケット呼び出しが完了したとき、指定された出口ルーチンを駆動します。

制約事項**:** この方式は、TCP/IP for z/VSE ではサポートされません。

どちらの場合も、関数が完了するのは、通知が送達されたときです。通知はいつでも送達される可能性があることに注意してください。場合によっては、アプリケーションが EZASMI マクロ呼び出しから戻される制御を受け取るよりも前に送達されることさえあります。従って、アプリケーションは、EZASMI マクロ呼び出しを発行するとすぐに、通知を処理する準備ができていることが重要です。

EZASMI マクロを使用して、APITYPE パラメーターを指定できます。APITYPE=2 のみが、サポートされている (デフォルトでもある) タイプです。これを指定すると、1 つのソケット記述子当たり複数の、未解決の非同期のソケット呼び出し (例えば、RECV および SEND 呼び出し) が可能になります。非同期のマクロ呼び出しが使用される場合は、ECB 方式が必要です。

非同期呼び出しのための ECB 入力パラメーターは、160 バイトのストレージ域を示さなければなりません。

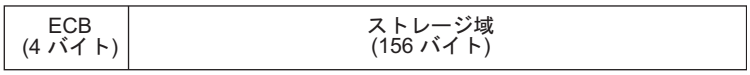


図 *19. ECB* 入力パラメーター

非同期関数処理中には、ECB に続く 156 バイトのストレージ域が使用されます。非同期関数呼び出しが完了するまで (つまり、ECB がポストされるまで)、アプリケーション・プログラムはこのストレージ域を変更してはなりません。

非同期関数は、次のようなシーケンスで処理されます。

1. アプリケーションは、ASYNC='ECB' を指定して EZASMI TYPE=INITAPI を発行する必要があります。ASYNC パラメーターは、この接続には最終的には非同期処理が使用されることを API に通知します。
2. ECB を伴う非同期関数がアプリケーションによって発行されると、その要求は処理のために待ち行列に入れられ、API は即時にアプリケーションに制御を戻します。関数が正常に待ち行列に入れられると、RETCODE=0 で戻り、ERRNO は EINPROGRESS に設定されます。関数のキューイングの途中でエラー条件が発生した場合、API は RETCODE=-1 で戻り、ERRNO が非同期操作のエラー状況を示します。この場合も ECB はポストされます。
3. 関数が完了する (これは、関数呼び出しがアプリケーションに戻る前でも起こる可能性があります) と、ECB がポストされ、関数特有の戻り情報 (RETCODE) とエラー情報 (ERRNO) が戻されます。

次の例は、非同期マクロ関数のコーディング例を示します。

```
EZASMI TYPE=READ,     READ A BUFFER OF DATA FROM THE                *
      S=SOCKNO,       CONNECTION PEER.  I MAY NEED TO               *
      NBYTES=COUNT,   WAIT SO GIVE CONTROL BACK TO ME               *
```

```
                BUF=DATABUF,    AND LET ME ISSUE MY OWN WAIT.                  *
                ERRNO=ERROR,    IT COULD BE PART OF A WAIT WHICH               *
                RETCODE=RCODE,  WOULD INCLUDE OTHER EVENTS.                    *
                ECB=MYECB,      SPECIFY ECB/STORAGE AREA FOR INTERFACE         *
                ERROR=ERRORRTN

          WAIT  MYECB           TELL VSE TO WAIT UNTIL READ IS DONE
```

エラー・メッセージおよび戻りコード

エラー・メッセージについては、「*IBM VSE/ESA* メッセージとコード 第 *1* 巻」および「*TCP/IP for VSE Messages*」を参照してください。

TCP/IP によって戻されるエラー・コードについては、83 ページの『ERRNO 値』を参照してください。

デバッグ

615 ページの『付録 B. EZASMI および EZASOKET インターフェースのデバッグ機能 (EZAAPI トレース)』を参照してください。

アセンブラー・プログラム用のマクロ

このセクションでは、この API に含まれる各マクロについて、説明、構文、パラメーター、およびその他の関連情報を説明します。

ACCEPT

ACCEPT マクロは、サーバーがクライアントから接続要求を受信したときに発行されます。

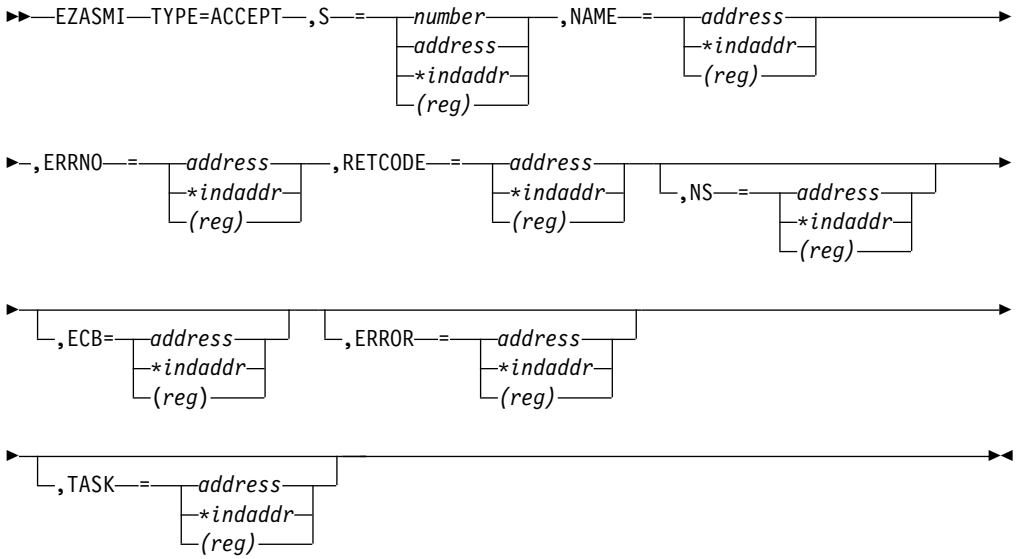
ACCEPT は、前もって SOCKET マクロによって作成されて LISTEN マクロによってマークされたソケットを指します。いくつかのピア・プロセスからの接続要求の完了を待機するようなプロセスでは、後にある ACCEPT マクロは、CONNECT マクロの 1 つが完了するまでブロックする可能性があります。これを回避するには、CONNECT マクロと ACCEPT マクロの間に SELECT マクロを発行します。並行サーバー・プログラムは、ACCEPT マクロを使用して、接続要求をサブタスクに渡します。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

発行されると、ACCEPT マクロは次のことを行います。

1. 保留中接続の待ち行列の最初の接続を受け入れます。
2. マクロで使われたソケットと同じプロパティーを持つ新規ソケットを作成し、それ以降のサーバー・マクロで使用できるようにクライアントのアドレスを戻します。新規ソケットは新しい接続を受け入れるために使用できるのではなく、呼び出し側プログラムによってそれ自身の接続のために使用できます。元のソケットは、呼び出し側プログラムがさらに接続要求を行うために使用可能のままです。
3. 呼び出し側プログラムに新規ソケット記述子を戻します。

フォーマット



パラメーター

S　入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、そこから接続を受け入れるソケットの記述子を指定します。

NAME

最初は、IPv4 または IPv6 アプリケーションが IPv4 または IPv6 ソケット・アドレス構造体へのポインターを設定します。呼び出しの完了時には、その構造体には、接続ピアのソケット・アドレスが挿入されます。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

FAMILY

ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT

ハーフワード 2 進フィールドであり、クライアントのポート番号を指定します。ネットワーク・バイト・オーダーのポート番号に設定されます。例えば、ポート番号が 10 進数で 5000 の場合、X'1388' に設定されます。

IPv4-ADDRESS

フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

NAMELEN
IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、クライアントのポート番号に設定されます。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。RETCODE が正の場合、RETCODE は新規ソケット番号です。

RETCODE が負の場合、ERRNO のエラー番号をチェックしてください。

NS TCP/IP for z/VSE にはサポートされていません。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。

- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

BIND

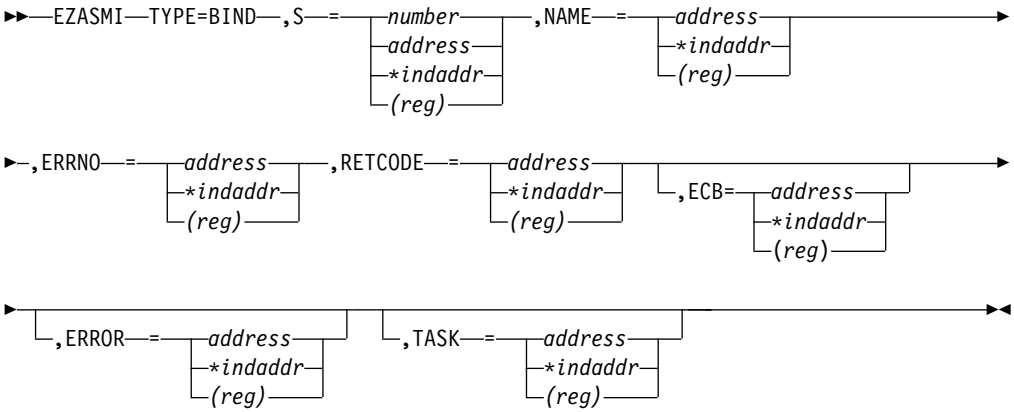
サーバー・プログラムでは、一般的に、BIND マクロは SOCKET マクロの後に続き、新規ソケットの作成プロセスを完了します。

BIND マクロでは、ポートを指定したり、システムによるポートの選択を許可したりできます。リスナー・プログラムを常に同じウェルノウン・ポートにバインドする必要があります。このようにすることで、クライアントは、CONNECT マクロを発行する際にどのソケット・アドレスを使用するのかを把握することができます。

AF_INET ドメインでは、ストリーム・ソケットに対する BIND マクロは、どのネットワークから接続要求を受け入れるのかを指定できます。アプリケーションは、そこから接続要求を受け入れるネットワークの IP アドレスを **ADDRESS** に設定することによって、ネットワーク・インターフェースを選択できます。または、アドレス・フィールドにフルワードのゼロを設定することによって、アプリケーションはどのネットワークからも接続要求を受け入れることができます。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

フォーマット



パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、ソケット記述子を指定します。

NAME

IPv4 または IPv6 アプリケーションは、IPv4 または IPv6 ソケット・アドレス構造体へのポインターを設定します。この構造体は、アプリケーションが接続を受け入れることができるポート番号と、IPv4 または IPv6 IP アドレスを指定します。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

FAMILY

ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT

ハーフワード 2 進数フィールドであり、クライアントのポート番号を指定します。ポート番号をゼロに設定すると、TCP/IP がポートを割り当てます。アプリケーションは、BIND マクロの後に GETSOCKNAME マクロを呼び出して、割り当てられたポートを知ることができます。

IPv4-ADDRESS

フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED

2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

NAMELEN

IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY

IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT

ハーフワード 2 進数フィールドであり、クライアントのポート番号に設定されます。ポート番号をゼロに設定すると、TCP/IP がポー

トを割り当てます。アプリケーションは、BIND マクロの後に GETSOCKNAME マクロを呼び出して、割り当てられたポートを知ることができます。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

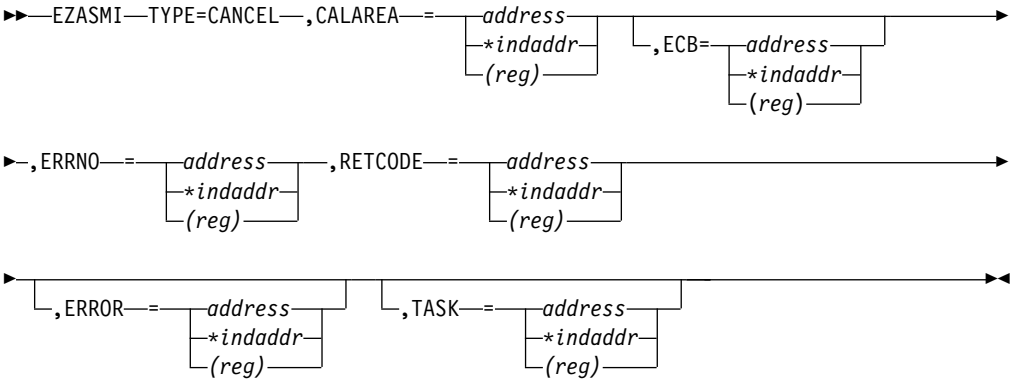
TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

CANCEL

CANCEL 関数は、進行中の呼び出しを終了します。

取り消される呼び出しには、ECB が指定されていなければなりません。

フォーマット



パラメーター

CALAREA

入力パラメーター。取り消される呼び出しに指定された ECB。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注: このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

出力パラメーター。フルワード 2 進数フィールド。RETCODE が 0 の場合、CANCEL は成功しました。取り消された呼び出しのエラー状況 (ERRNO) が ECANCELED に設定されます。RETCODE が -1 の場合、CANCEL は失敗しました。ERRNO のエラー・コードをチェックしてください。例えば、選択された要求が進行中のために取り消すことができない場合、ERRNO は EINPROGRESS に設定され、選択された要求が既に完了しているために取り消すことができない場合、EINVAL に設定されます。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

CLOSE

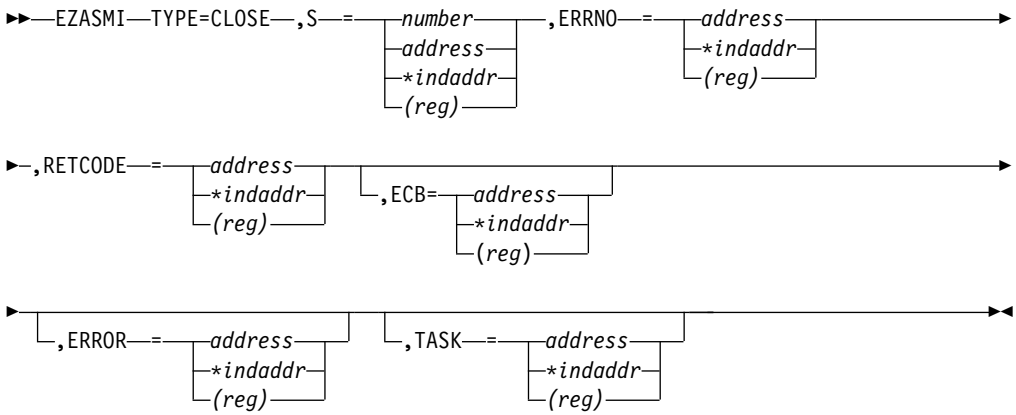
CLOSE マクロはソケットをシャットダウンし、そのソケットに割り振られているリソースを解放します。

CLOSE マクロを発行する前に、SHUTDOWN マクロを発行してください。

CLOSE は、並行サーバーによっても発行されます。その場合、サーバーがサブタスク・プログラムにソケットを与えた後に発行されます。並行サーバーは、GIVESOCKET を発行し、子クライアントが正常に TAKESOCKET を発行したという通知を受け取った後、CLOSE マクロを発行して所有権の引き渡しを完了します。

注: 待ち行列に入力データまたは出力データがあるときにストリーム・ソケットがクローズされると、ストリーム接続はリセットされ、データ伝送が不完全になることがあります。SETSOCKET を使用して SO_LINGER 条件を設定することができ、そうすると、TCP/IP は、CLOSE マクロが発行された後、指定された期間は引き続きデータ送信を行います。SO_LINGER について詳しくは、 425 ページの『SETSOCKOPT』を参照してください。

フォーマット



パラメーター

S　入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、クローズされるソケットを指定します。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

CONNECT

CONNECT マクロは、ローカル・ソケットとリモート・ソケットの間に接続を確立するためにクライアントが使用します。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

ストリーム・ソケットについては、CONNECT マクロは次のことを行います。

- 以前に BIND が発行されていない場合、ストリーム・ソケットのバインド処理を完了します。
- リモート・ソケットへの接続を試みます。この接続が完了しなければ、データは転送できません。

データグラム・ソケットの場合、CONNECT は必須ではありませんが、使用すると、宛先を指定せずにメッセージを送信できます。

どちらのタイプのソケットでも、CONNECT マクロは以下のシーケンスで使用されます。

1. サーバーは、BIND および LISTEN (ストリーム・ソケットのみ) を発行して、受動オープン・ソケットを作成します。
2. クライアントは、CONNECT を発行して、接続を要求します。
3. サーバーは、受動オープン・ソケットで接続を受け入れて、接続された新規ソケットを作成します。

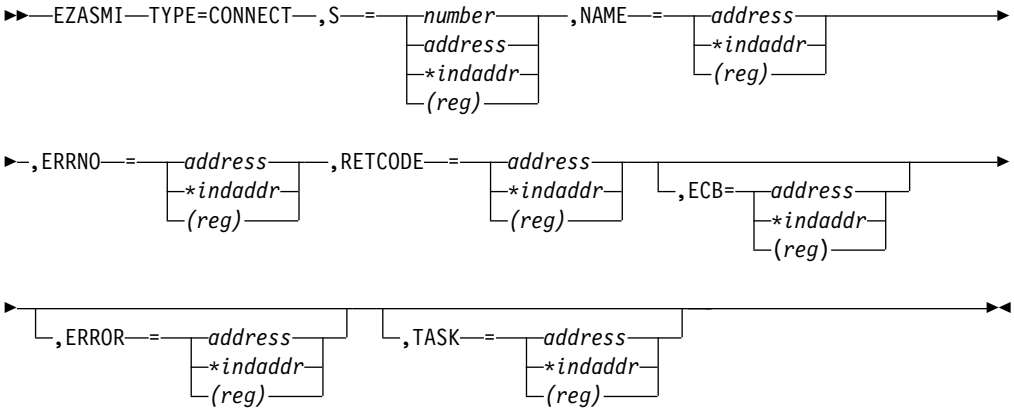
ソケットがブロッキング・モードの場合、CONNECT は、接続が確立するまで、または、エラーを受け取るまで、呼び出し側プログラムをブロックします。

ソケットが非ブロッキング・モードの場合、接続要求の成功を戻りコードが示します。

- ゼロの RETCODE は、接続が完了したことを示します。
- RETCODE がゼロ以外で ERRNO が EINPROGRESS の場合、接続を完了できなかったことを示しますが、ソケットが非ブロッキングであるため CONNECT マクロは処理を完了します。

呼び出し元は、SELECT の呼び出しと、ソケットへの書き込みテストを行うことによって、接続セットアップの完了をテストする必要があります。完了は、2 回目の CONNECT を発行することではチェックできません。

フォーマット



パラメーター

S　入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、ソケット記述子を指定します。

NAME

入力パラメーター。CONNECT の NAME パラメーターは、ローカルのクライアント・ソケットが接続される、ターゲットの IPv4 または IPv6 ソケット・アドレスを指定します。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

FAMILY

ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、クライアントのポート番号を指定します。

IPv4-ADDRESS
フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

NAMELEN
IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、クライアントのポート番号に設定されます。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

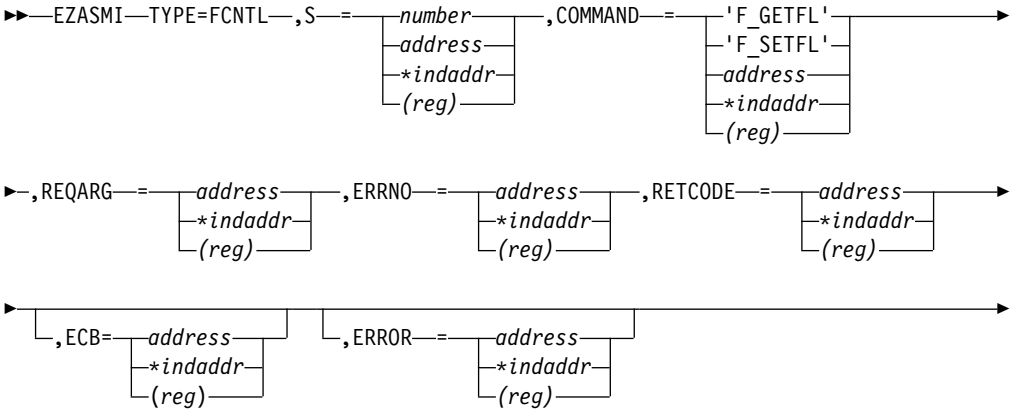
FCNTL

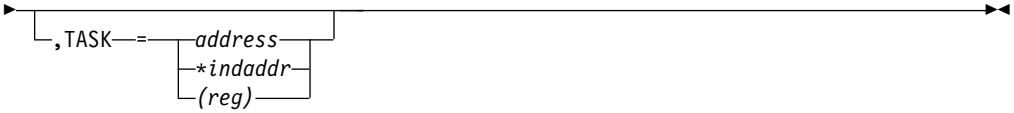
ソケットのブロッキング・モードは、FNDELAY フラグを使用して、照会するか、または非ブロッキングに設定することができます。

FNDELAY フラグは、プログラム内に定義されていなくても、照会または設定することができます。

ソケットのブロッキングを制御する別の方法については、 399 ページの『IOCTL』を参照してください。

フォーマット





パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、非ブロッキングに設定するか照会するソケットの、ソケット記述子を指定します。

COMMAND
入力パラメーター。フルワード 2 進数フィールドまたはリテラルであり、FNDELAY フラグを以下のいずれかの値に設定します。

3 または **'F_GETFL'**
ソケットのブロッキング・モードを照会します。

4 または **'F_SETFL'**
ソケットのモードを非ブロッキングに設定します。REQARG は TCP/IP によって設定されます。

FNDELAY フラグは、ソケットを非ブロッキング・モードに設定します。ブロックする可能性のある呼び出し (READ、READV、および RECV) にデータが存在しない場合、呼び出しは -1 を戻し、ERRNO は EWOULDBLOCK に設定されます。

REQARG
フルワード 2 進数フィールドであり、TCP/IP が FNDELAY フラグを設定するのに使用するマスクを含みます。

- COMMAND が 3 (照会) に設定されている場合、REQARG フィールドは 0 に設定される必要があります。
- COMMAND が 4 (設定) に設定されている場合は、次のとおりです。
  - REQARG を 4 に設定すると、FNDELAY フラグはオンになります。これにより、ソケットは非ブロッキング・モードになります。
  - REQARG を 0 に設定すると、FNDELAY フラグはオフになります。これにより、ソケットはブロッキング・モードになります。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO フィールドは無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

- COMMAND が 3 (照会) に設定された場合、ビット・ストリングが戻されます。
  - RETCODE に X'00000004' が入っている場合、ソケットは非ブロッキングです。FNDELAY フラグはオンです。

  - RETCODE に X'00000000' が入っている場合、ソケットはブロッキングです。FNDELAY フラグはオフです。
- COMMAND フィールドが 4 (設定) だった場合、呼び出しが成功すると RETCODE にゼロが戻されます。どちらの COMMAND の場合でも、RETCODE -1 はエラーを示します。ERRNO のエラー番号をチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注: このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

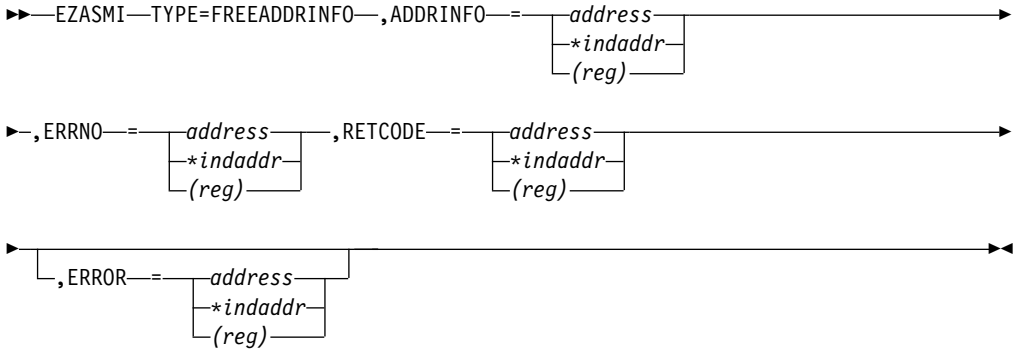
TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

FREEADDRINFO

FREEADDRINFO マクロは、RES パラメーターで GETADDRINFO により返されるすべてのアドレス情報構造体を解放します。

重要: この関数呼び出しは TCP/IP for z/VSE では使用できません。

フォーマット



パラメーター

ADDRINFO
入力パラメーター。TYPE=GETADDRINFO RES 引数によって返されるアドレス情報構造体セットのアドレス。

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO フィールドは無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ERROR

入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

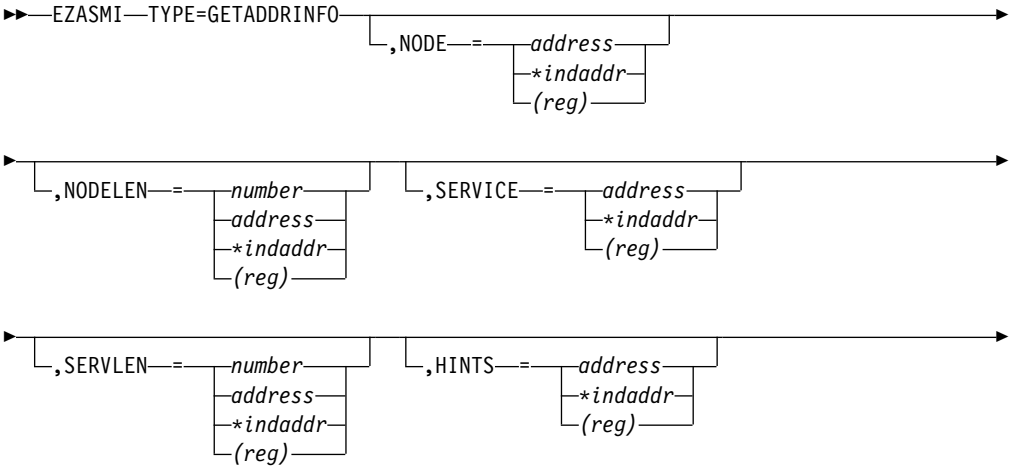
GETADDRINFO

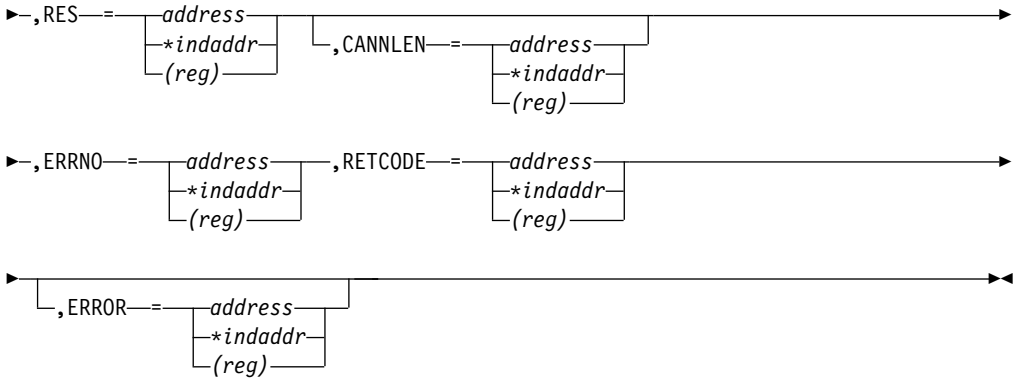
GETADDRINFO マクロは、サービス・ロケーションの名前 (例えば、ホスト名)、サービス名、またはその両方を、ソケット・アドレスのセットおよび他の関連情報に変換します。

この情報を使用して、ソケットを作成したり、指定したサービスに接続したり、またはそのサービスにデータグラムを送信したりできます。

重要**:** この関数呼び出しは TCP/IP for z/VSE では使用できません。

フォーマット





パラメーター

NODE

入力パラメーター。照会されるホスト名が入る、最大で 255 バイト長のストレージ。AI_NUMERICHOST フラグが、HINTS オペランドにより示されるストレージで指定されている場合、NODE には、ネットワーク・バイト・オーダー表示形式の、照会されるホストの IP アドレスが入らなければなりません。これはオプションのフィールドですが、指定した場合は NODELEN もコーディングする必要があります。照会される NODE 名は、NODELEN まで、または最初の 2 進ゼロまでで構成されます。

スコープ情報をホスト名に、*node%scope information* という形式を用いて追加できます。結合される情報は、255 バイト以下でなければなりません。

NODELEN

入力パラメーター。フルワード 2 進数フィールドであり、NODE フィールドに指定されたホスト名の長さに設定されます。余分のブランクを含めることはできません。これはオプションのフィールドですが、指定した場合は NODE もコーディングする必要があります。

SERVICE

入力パラメーター。照会されるサービス名が入る、最大で 32 バイト長のストレージ。AI_NUMERICSERV フラグが、HINTS オペランドにより示されるストレージで指定されている場合、SERVICE には、表示形式の、照会されるポート番号が入る必要があります。これはオプションのフィールドですが、指定した場合は SERVLEN もコーディングする必要があります。照会される SERVICE 名は、SERVLEN まで、または最初の 2 進ゼロまでで構成されます。

SERVLEN

入力パラメーター。フルワード 2 進数フィールドであり、SERVICE フィールドに指定されたサービス名の長さに設定されます。余分のブランクを含めることはできません。これはオプションのフィールドですが、指定した場合は SERVICE もコーディングする必要があります。

HINTS

入力パラメーター。HINTS 引数を指定する場合、それには入力値が含まれる addrinfo 構造体のアドレスが入ります。その入力値によって、オプションを指定したり、返される情報を特定のソケット・タイプ、アドレス・ファミリー、またはプロトコルに制限したりすることで、操作を指示できます。

HINTS 引数を指定しない場合、返される情報は、FLAGS、SOCTYPE、および PROTO フィールドには値 0、AF フィールドには値 AF_UNSPEC が含まれる構造体を参照した場合と同じものになります。

アドレス情報構造体には、以下のフィールドがあります。

FLAGS

フルワード 2 進数フィールド。ビット単位の値 0、または以下のいずれか 1 つ以上の値が指定されなければなりません。

AI-PASSIVE (X'00000001') または **10** 進数値の **1**。

返される RES により示される NAME に挿入する方法を指定します。

このフラグを指定した場合、返されるアドレス情報は、指定したサービス (例えば、BIND 呼び出し) の着信接続を受け入れるソケットのバインドでの使用に適しています。この場合、NODE 引数が指定されていないと、返される RES により示されるソケット・アドレス構造体の IP アドレス部分は、IPv4 アドレスの場合は INADDR_ANY、IPv6 アドレスの場合は IPv6 未指定アドレス (in6addr_any) に設定されます。

このフラグが設定されていない場合、返されるアドレス情報は、CONNECT 呼び出し (コネクション型プロトコルの場合)、または CONNECT、SENDTO、SENDMSG 呼び出し (コネクションレス・プロトコルの場合) に適します。この場合、NODE 引数が指定されていないと、返される RES により示されるソケット・アドレス構造体の IP アドレス部分は、IPv4 アドレスの場合はデフォルトのループバック・アドレス (127.0.0.0)、IPv6 アドレスの場合はデフォルトのループバック・アドレス (::1) に設定されます。

このフラグは、NODE 引数が指定されている場合は無視されます。

AI-CANONNAMEOK (X'00000002') または **10** 進数値の **2**。

このフラグが指定され、かつ NODE 引数が指定されている場合、GETADDRINFO 呼び出しは、NODE 引数に対応する正規名を決定しようとします。

AI-NUMERICHOST (X'00000004') または **10** 進数値の **4**。

このフラグを指定した場合は、NODE 引数は、表示形式の数値ホスト・アドレスでなければなりません。そうしない場合、ホストが見つからない [EAI_NONAME] というエラーが戻されます。

AI-NUMERICSERV (X'00000008') または **10** 進数値の **8**。

このフラグを指定した場合は、SERVICE 引数は、表示形式の数値ポートでなければなりません。そうしない場合、[EAI_NONAME] というエラーが戻されます。

AI-V4MAPPED (X'00000010') または **10** 進数値の **16**。

このフラグを指定した場合に、AF フィールドに値

AF_INET6、または IPv6 がサポートされている場合には値 AF_UNSPEC を指定すると、呼び出し元は IPv4 マップの IPv6 を受け入れます。さらに、AI-ALL フラグが指定されていない場合、IPv6 アドレスが検出されないと、照会は IPv4 アドレスを対象として実行されます。IPv4 アドレスが検出された場合、それらは IPv4 マップの IPv6 アドレスとして返されます。

AF フィールドに値 AF_INET6 が入っていないか、または AF フィールドに AF_UNSPEC が入っているものの IPv6 がシステムでサポートされていない場合、このフラグは無視されます。

AI-ALL (X'00000020') または **10** 進数値の **32**。
AF フィールドに値 AF_INET6 が入っており、かつ AI-ALL が設定されている場合、さらに AI-V4MAPPED フラグを、すべてのアドレス (IPv6 および IPv4 マップの IPv6 アドレス) の受け入れを呼び出し元に指示するように設定する必要があります。システムが IPv6 をサポートする場合に AF フィールドに値 AF_UNSPEC が入っており、かつ AI-ALL が設定されている場合、呼び出し元は、IPv6 アドレスと、IPv4 アドレス (AI-V4MAPPED が設定されていない場合) または IPv4 マップの IPv6 アドレス (AI-V4MAPPED が設定されている場合) のいずれかを受け入れます。まず IPv6 アドレスについて照会が行われ、それが正常に実行されると、IPv6 アドレスが返されます。次いで別の照会が IPv4 アドレスについて行われ、検出されたすべての IPv4 アドレスも、IPv4 マップの IPv6 アドレス (AI-V4MAPPED も指定されている場合) または IPv4 アドレス (AI-V4MAPPED が指定されていない場合) のいずれかとして返されます。

AF フィールドに値 AF_INET6 が入っていないか、またはシステムが IPv6 をサポートしているが値 AF_UNSPEC が入っていない場合、このフラグは無視されます。

AI-ADDRCONFIG (X'00000040') または **10** 進数値の **64**。
このフラグが指定されている場合、リゾルバーが以下のいずれかが真であると判断すると、NODE にある名前に対する照会が行われます。

- システムが IPv6 対応であり、少なくとも 1 つの IPv6 インターフェースがある場合、リゾルバーは IPv6 (AAAA または A6 DNS) レコードの照会を行います。
- システムが IPv4 対応であり、少なくとも 1 つの IPv4 インターフェースがある場合、リゾルバーは IPv4 IPv4 (A DNS) レコードの照会を行います。ループバック・アドレスは、この場合には有効なインターフェースとは見なされません。

注: 上記のフラグのバイナリー OR を COBOL プログラムで実行する場合は、以下の例で示すとおり、必要な COBOL ステートメントを追加するだけで済みます。COBOL ADD の後の FLAGS フィールドの値が 10 進数 80 または X'00000050' であることに注意してください。これは AI_V4MAPPED および AI_ADDRCONFIG、または X'00000010' および X'00000040' での OR 実行の合計です。

```
01 AI-V4MAPPED PIC 9(8) BINARY VALUE 16.
01 AI-ADDRCONFIG PIC 9(8) BINARY VALUE 64.

ADD AI-V4MAPPED TO FLAGS.
ADD AI-ADDRCONFG TO FLAGS.
```

AF フルワード 2 進数フィールド。返される情報を特定のアドレス・ファミリーに制限するために使用します。値 AF_UNSPEC は、呼び出し元がどのプロトコル・ファミリーでも受け入れることを意味します。10 進数の値 0 は AF_UNSPEC を示します。10 進数の値 2 は AF_INET を示し、10 進数の値 19 は AF_INET6 を示します。

SOCTYPE

フルワード 2 進数フィールド。返される情報を特定のソケット・タイプに制限するために使用します。値 0 は、呼び出し元がどのソケット・タイプでも受け入れることを意味します。特定のソケット・タイプを指定しない場合 (例えば、値 0)、サポートされるすべてのソケット・タイプに関する情報が返されます。以下に示すのは、受け入れ可能なソケット・タイプです。

タイプ名	**10** 進数値	説明
SOCK_STREAM	1	ストリーム・ソケット用
SOCK_DGRAM	2	データグラム・ソケット用
SOCK_RAW	3	ロー・プロトコル・インターフェース用

これ以外の値では失敗し、戻りコード EAI_SOCTYPE が出されます。SOCK_RAW は受け入れられますが、SERVICE が数値 (例えば、SERVICE=23) の場合にのみ有効であることに注意してください。SERVICE 名の検索は、SOCK_STREAM または SOCK_DGRAM 以外のいずれかのプロトコル値を使用した適切なサービス・ファイルでは、決して行われません。

PROTO が 0 ではなく、かつ SOCTYPE が 0 である場合、PROTO の受け入れ可能な入力値は、IPPROTO_TCP および IPPROTO_UDP のみです。それ以外の場合、GETADDRINFO 呼び出しは失敗し、戻りコード EAI_BADFLAGS が出されます。

SOCTYPE および PROTO の両方が 0 と指定されている場合、GETADDRINFO は以下のように進みます。

- SERVICE が NULL の場合、または SERVICE が数値の場合、返される addrinfo により、SOCTYPE の指定はデフォルトの SOCK_STREAM になります。

- SERVICE がサービス名として指定されている場合 (例えば、SERVICE=FTP)、GETADDRINFO 呼び出しは適切なサービス・ファイルを 2 回検索します。最初の検索では SOCK_STREAM がプロトコルとして使用され、2 番目の検索では SOCK_DGRAM がプロトコルとして使用されます。この場合には、デフォルトのソケット・タイプのプロビジョンはありません。

SOCTYPE および PROTO の両方が非ゼロに指定されている場合、それらは SERVICE で指定された値にかかわらず、互換性がなければなりません。このコンテキストでは、互換とは以下のいずれかを意味します。

- SOCTYPE=SOCK_STREAM および PROTO=IPPROTO_TCP
- SOCTYPE=SOCK_DGRAM および PROTO=IPPROTO_UDP
- SOCTYPE が SOCK_RAW と指定されている場合、PROTO には任意の値が指定可能

PROTO

フルワード 2 進数フィールド。返される情報を特定のプロトコルに制限するために使用します。値 0 は、呼び出し元がどのプロトコルでも受け入れることを意味します。以下に示すのは、受け入れ可能なプロトコルです。

プロトコル名	**10** 進数値	説明
IPPROTO_TCP	6	TCP
IPPROTO_UDP	17	ユーザー・データグラム

SOCTYPE が 0 で、かつ PROTO が非ゼロである場合、PROTO の受け入れ可能な入力値は IPPROTO_TCP および IPPROTO_UDP のみです。それ以外の場合、GETADDRINFO 呼び出しは失敗し、戻りコード EAI_BADFLAGS が出されます。

PROTO および SOCTYPE の両方が 0 と指定されている場合、GETADDRINFO は以下のように進みます。

- SERVICE が NULL の場合、または SERVICE が数値の場合、返される addrinfo により、SOCTYPE の指定はデフォルトの SOCK_STREAM になります。
- SERVICE がサービス名として指定されている場合 (例えば、SERVICE=FTP)、GETADDRINFO は適切なサービス・ファイルを 2 回検索します。最初の検索では SOCK_STREAM がプロトコルとして使用され、2 番目の検索では SOCK_DGRAM がプロトコルとして使用されます。この場合には、デフォルトのソケット・タイプのプロビジョンはありません。

PROTO および SOCTYPE の両方が非ゼロに指定されている場合、それらは SERVICE で指定された値にかかわらず、互換性がなければなりません。このコンテキストでは、互換とは以下のいずれかを意味します。

- SOCTYPE=SOCK_STREAM および PROTO=IPPROTO_TCP

- SOCTYPE=SOCK_DGRAM および PROTO=IPPROTO_UDP
- SOCTYPE=SOCK_RAW、この場合には PROTO に任意の値が指定可能

SERVICE に指定された値の検索が失敗した場合 (例えば、サービス名が入力プロトコルを使用する適切なサービス・ファイルに存在していない場合)、GETADDRINFO 呼び出しは失敗し、戻りコード EAI_SERVICE が出されます。

NAMELEN
フルワード 2 進数フィールド。入力では、このフィールドは 0 であることが必要です。

CANNONNAME
フルワード 2 進数フィールド。入力では、このフィールドは 0 であることが必要です。

NAME
フルワード 2 進数フィールド。入力では、このフィールドは 0 であることが必要です。

NEXT フルワード 2 進数フィールド。入力では、このフィールドは 0 であることが必要です。

RES 最初はフルワード 2 進数フィールド。成功して戻ると、このフィールドには 1 つ以上のアドレス情報構造体のチェーンへのポインターが入ります。この構造体は、呼び出し側アプリケーションのキーに割り振られます。複数の MVS™ タスク間で、これらの構造体を使用したり参照したりしないでください。構造体の使用を終了した場合は、返されたポインターを FREEADDRINFO 呼び出しで指定することで、そのストレージを明示的に解放します。明示的に解放されていないストレージは、タスクの終了時に解放されます。

アドレス情報構造体には、以下のフィールドがあります。

FLAGS
フルワード 2 進数フィールドであり、出力としては使用されません。

AF フルワード 2 進数フィールド。このフィールドで返される値は、返されたアドレス NAME での使用に適したソケットを作成するために、TYPE=SOCKET マクロの AF= 引数として使用できます。

SOCTYPE
フルワード 2 進数フィールド。このフィールドで返される値は、返されたアドレス NAME での使用に適したソケットを作成するために、TYPE=SOCKET マクロの SOCTYPE= 引数として使用できます。

PROTO
フルワード 2 進数フィールド。このフィールドで返される値は、返されたアドレス NAME での使用に適したソケットを作成するために、TYPE=SOCKET マクロの PROTO= 引数として使用できます。

NAMELEN
フルワード 2 進数フィールド。NAME ソケット・アドレス構造体の長さ。このフィールドで返される値は、AI_PASSIVE フラグに従って、そのようなソケットがある TYPE=CONNECT または TYPE=BIND マクロの引数として使用できます。

CANNONNAME
フルワード 2 進数フィールド。NODE によって指定された値の正規名が入るストレージのアドレス。最初は、このフィールドは 0 でなければなりません。NODE 引数が指定され、AI_CANONNAMEOK フラグが HINTS 引数によって指定された場合、最初に返されたアドレス情報構造体の CANONNAME フィールドには、入力 NODE 引数に対応する正規名が入ったストレージのアドレスが入ります。正規名が使用不可の場合、CANONNAME は、NODE 引数、または同じ内容のストリングを参照します。CANNLEN フィールドには、返される正規名の長さが入ります。

NAME
フルワード 2 進数フィールド。戻されたソケット・アドレス構造体のアドレス。このフィールドで返される値は、AI_PASSIVE フラグに従って、そのようなソケットがある TYPE=CONNECT または TYPE=BIND マクロの引数として使用できます。

NEXT フルワード 2 進数フィールド。リスト上の次のアドレス情報構造体のアドレスが入るか、それがリスト上の最後の構造体の場合は 0 が入ります。

CANNLEN
最初は入力パラメーター。RES CANONNAME フィールドにより返される正規名の長さを入れるために使用する、フルワード 2 進フィールド。これはオプションのフィールドです。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO フィールドは無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

GETCLIENTID

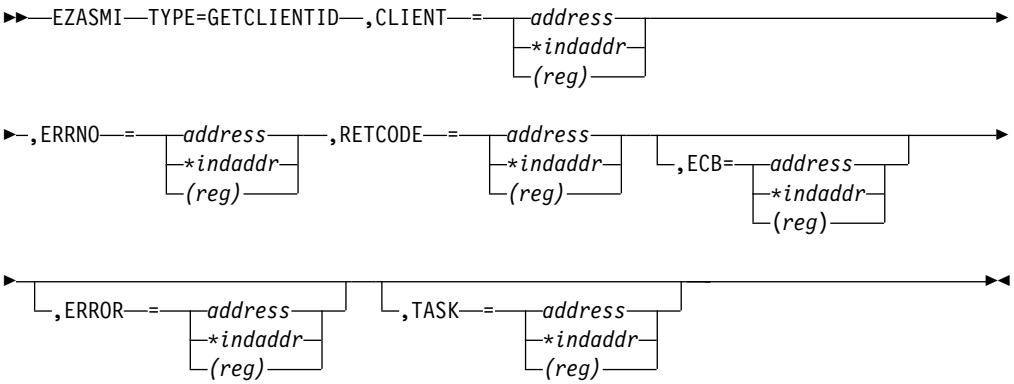
GETCLIENTID マクロは、呼び出し側のアプリケーションが TCP/IP アドレス・スペースに認識される ID を戻します。

戻されたクライアント ID 構造体は、GIVESOCKET および TAKESOCKET マクロで使用されます。

重要: IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

GETCLIENTID がサーバーまたはクライアントによって呼び出されると、呼び出し元アプリケーションの ID が戻されます。

フォーマット



パラメーター

CLIENT

呼び出しを発行したアプリケーションを記述する、クライアント ID 構造体です。

DOMAIN

フルワード 2 進数であり、クライアントのドメインを指定します。入力では、これは AF_INET のオプション・パラメーターであり、クライアントのドメインを指定するための AF_INET6 には必須パラメーターです。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2、または AF_INET6 を示す 10 進数の 19 に設定されます。出力では、これはクライアントの返されたドメインです。

NAME

8 バイトの文字フィールドです。パーティション ID から構成され、左寄せでブランクを入れます。

TASK

8 バイトの文字フィールドです。このタスク ID は、INITAPI 呼び出しによってユーザーが指定するか、システムでデフォルト設定されます (詳しくは、INITAPI 呼び出しの説明を参照してください)。

RESERVED
20 バイトの予約済み文字フィールドを指定します。このフィールドは必須であり、TCP/IP によって内部的に使用されます。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

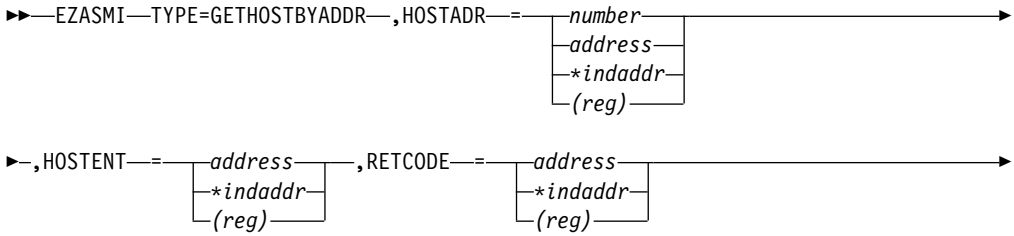
TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

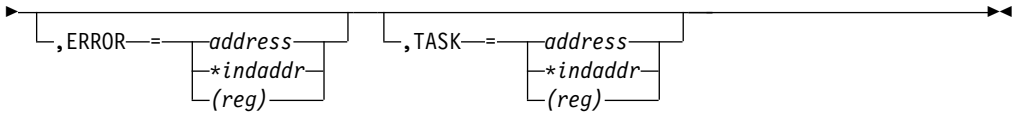
GETHOSTBYADDR

GETHOSTBYADDR マクロは、このマクロで指定された IP アドレスを持つホストの、ドメイン・ネームおよび別名を戻します。

1 つの TCP/IP ホストは、複数の別名および IP アドレスを持つことができます。

フォーマット





注: GETHOSTBYADDR および GETHOSTBYNAME は、いずれも同じ静的領域を使用して HOSTENT 構造体を戻します。この静的領域は、これらの関数のどちらかが次に同じスレッドで呼び出されるまで、または TERMAPI までしか有効でありません。

パラメーター

HOSTADR
入力パラメーター。フルワードの符号なし 2 進数フィールドであり、名前を検出するホストの IP アドレスに設定されます。

HOSTENT
入力パラメーター。フルワードのフィールドであり、このマクロで戻される、HOSTENT 構造体のアドレスを含みます。HOSTENT 構造体については、 363 ページの図 20 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
>0	呼び出しは成功しました。
-1	エラーが発生しました。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

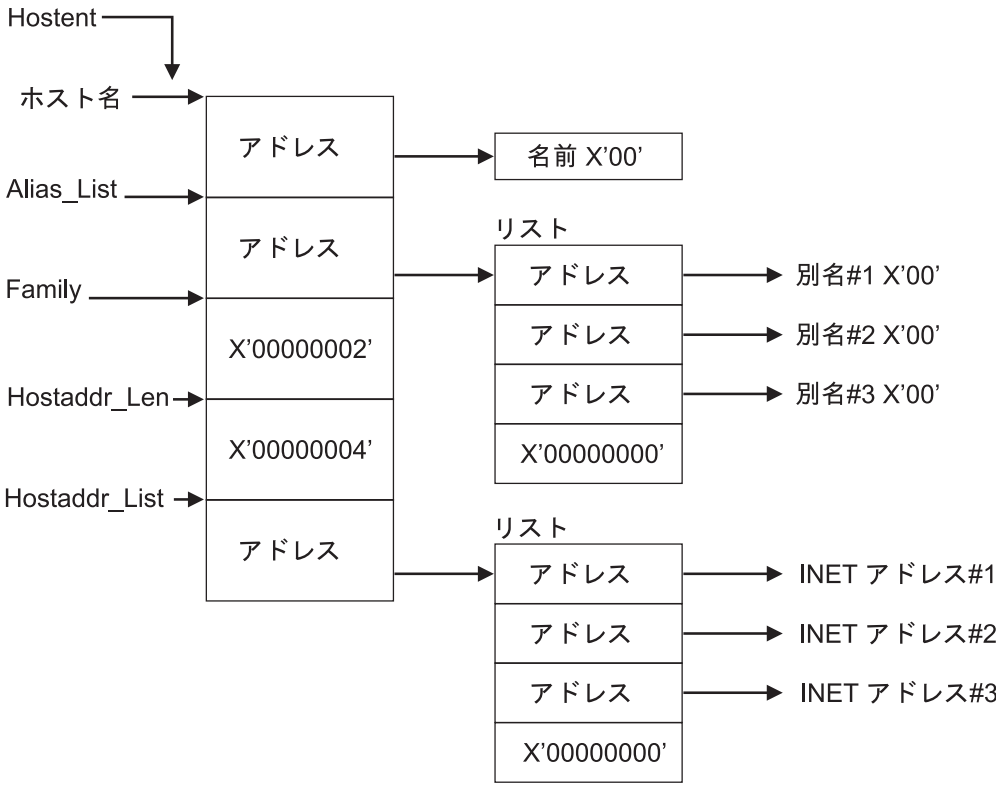


図 *20. GETHOSTBYADDR* マクロによって戻される *HOSTENT* 構造体

GETHOSTBYADDR は、図 20 に示す HOSTENT 構造体を戻します。この構造体に含まれるのは次のとおりです。

- マクロによって戻されるホスト名のアドレス。名前は可変長であり、X'00' で終わります。
- GETHOSTBYADDR によって戻される別名を指すアドレス・リストのアドレス。このリストは、ポインター X'00000000' で終わります。それぞれの別名は可変長フィールドであり、X'00' で終わります。

  注**:** 別名はサポートされていません。
- FAMILY フィールドに戻される値は、常に AF_INET を表す 2 です。
- HOSTADDR_LEN フィールドに戻されるホスト IP アドレスの長さは、常に AF_INET を表す 4 です。
- マクロによって戻されるホスト IP アドレスを指すアドレス・リストのアドレス。リストは、ポインター X'00000000' で終わります。

HOSTENT 構造体は、間接アドレッシングを使用して、可変の個数の別名および IP アドレスを戻します。

GETHOSTBYNAME

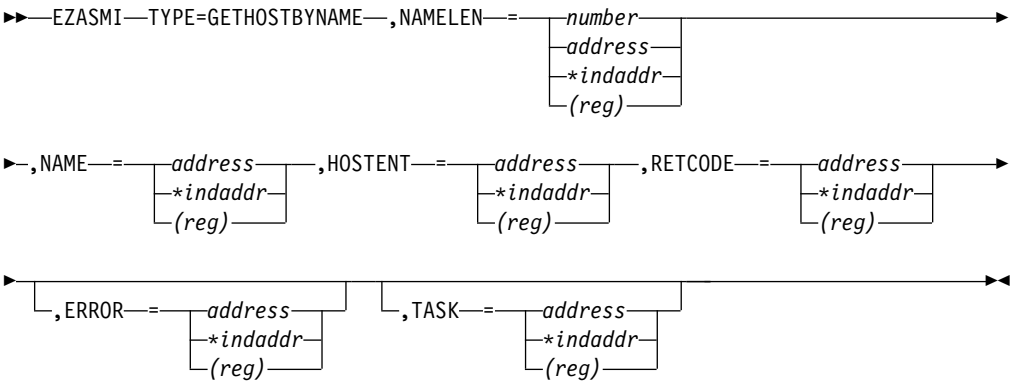
GETHOSTBYNAME マクロは、マクロに指定されたドメイン・ネームを持つホストの、別名および IP アドレスを戻します。

TCP/IP は、ネーム・サーバーが存在すれば、ネーム・サーバーを介してホスト名を解決しようとします。

シンボル名を IP アドレスに変換するために呼び出しが行われると、TCP/IP for z/VSE は、ローカル名テーブル (DEFINE NAME によって作成される) を最初に検索します。この検索が失敗した場合、名前は指定された DNS (SET DNSx で設定される) に渡されます。TCP/IP for z/VSE は、応答を受け取るか、すべてのサーバーにポーリングをし終わるまで、DNS1 から始めて各 DNS を試します。応答する最初のサーバーが、要求が成功か失敗かを決定します。ある DNS 内での検索が失敗した場合、デフォルトのドメイン・ストリング (SET DEFAULT_DOMAIN で指定される) が名前に (ピリオドの後に続けて) 付加され、ネーム解決のために最後にその DNS に情報が求められます。

ホスト名が見つからない場合、戻りコードは -1 になります。

フォーマット



注**:** GETHOSTBYADDR および GETHOSTBYNAME は、いずれも同じ静的領域を使用して **hostent** 構造体を戻します。この静的領域は、これらの関数のどちらかが次に同じスレッドで呼び出されるまで、または TERMAPI までしか有効でありません。

パラメーター

NAMELEN

入力パラメーター。フルワード 2 進数フィールドのアドレスまたは値であり、名前および別名フィールドの長さを指定します。この長さの最大値は 255 バイトです。

NAME

24 文字までの文字ストリングで、ホスト名に設定されます。この呼び出しは、この名前に対する HOSTENT のアドレスを戻します。

HOSTENT

出力パラメーター。フルワードのフィールドであり、このマクロで戻される、HOSTENT のアドレスを含みます。HOSTENT 構造体については、365 ページの図 21 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。

-1 エラーが発生しました。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

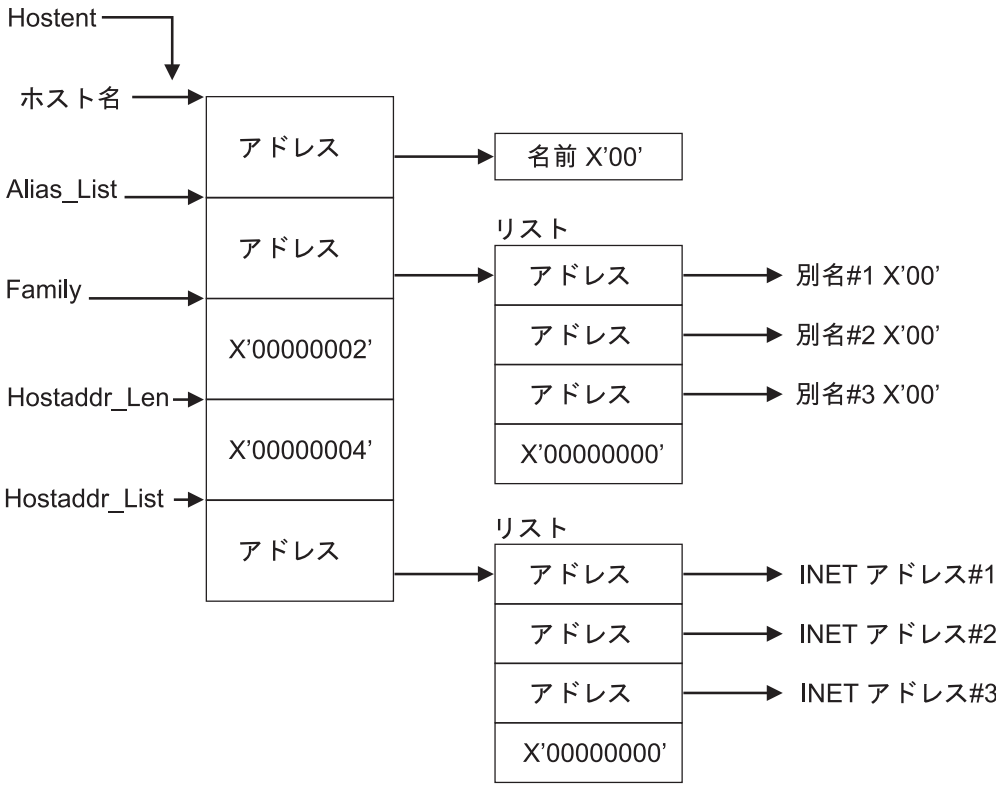


図 *21. GETHOSTBYNAME* マクロによって戻される *HOSTENT* 構造体

GETHOSTBYNAME は、図 21 に示す HOSTENT 構造体を戻します。この構造体に含まれるのは次のとおりです。

- マクロによって戻されるホスト名のアドレス。名前は可変長であり、X'00' で終わります。
- GETHOSTBYNAME によって戻される別名を指すアドレス・リストのアドレス。このリストは、ポインター X'00000000' で終わります。それぞれの別名は可変長フィールドであり、X'00' で終わります。

  注: 別名はサポートされていません。
- FAMILY フィールドに戻される値は、常に AF_INET を表す 2 です。
- HOSTADDR_LEN フィールドに戻されるホスト IP アドレスの長さは、常に AF_INET を表す 4 です。
- マクロによって戻されるホスト IP アドレスを指すアドレス・リストのアドレス。リストは、ポインター X'00000000' で終わります。

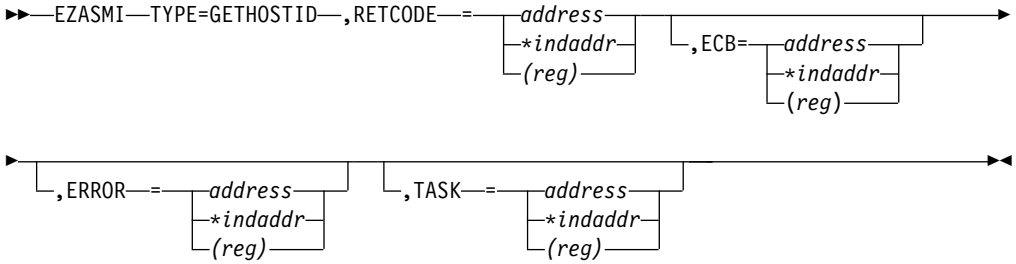
HOSTENT 構造体は、間接アドレッシングを使用して、可変の個数の別名および IP アドレスを戻します。

GETHOSTID

GETHOSTID マクロは、現行ホストの 32 ビットの ID を戻します。

この値は、デフォルトのホーム IP アドレスです。

フォーマット



パラメーター

RETCODE
出力パラメーター。ホストの 32 ビットの IP アドレスを戻します。RETCODE が -1 である場合、エラーがあったことを示します。このマクロには ERRNO パラメーターはありません。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注: このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

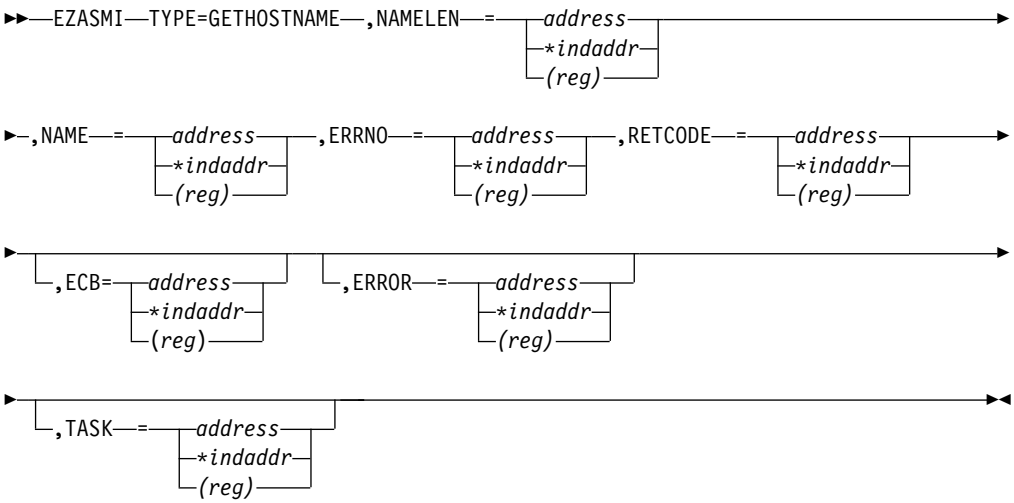
TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

GETHOSTNAME

GETHOSTNAME マクロは、このプログラムが実行されているホスト・プロセッサーの名前を戻します。

NAMELEN 文字までが NAME フィールドにコピーされます。

フォーマット



パラメーター

NAMELEN

入出力パラメーター。フルワードで、値、または、名前フィールドの長さが設定されたフルワード 2 進数フィールドのアドレスに設定されます。名前フィールドに指定できる最大長は、255 文字です。

NAME

最初は、アプリケーションがホスト名を受け取るフィールドへのポインターを設定します。TCP/IP for z/VSE で許容される最大長は 64 文字です。呼び出しの完了時には、このフィールドには、**NAMELEN** に長さが戻されるホスト名の長さが入ります。

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

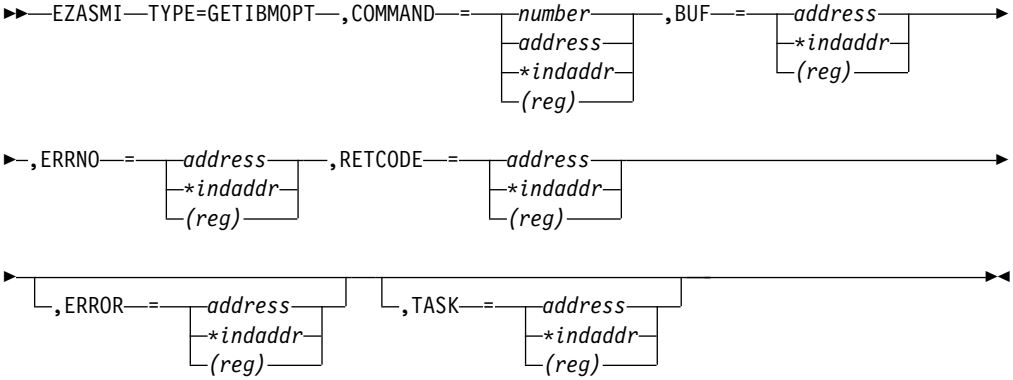
GETIBMOPT

GETIBMOPT マクロは、特定の z/VSE システムにインストールされている TCP/IP イメージの数、各イメージの状況、バージョン、および名前を返します。

この情報に基づき、呼び出し側は INITAPI マクロを使用して、接続先とする TCP/IP イメージを動的に選択できます。

重要**:** この関数呼び出しは TCP/IP for z/VSE では使用できません。

フォーマット



パラメーター

COMMAND
入力パラメーター。処理するコマンドを指定する値またはフルワード 2 進数のアドレス。有効な値は 1 のみです。

BUF 出力パラメーター。アクティブな各 TCP/IP イメージの状況、バージョン、および名前の格納先の 100 バイト・バッファーです。正常終了して返された場合、これらのバッファーのエントリーには、最大 8 つのアクティブな TCP/IP イメージの状況、名前、およびバージョンが入っています。次の表に、呼び出し完了時の BUF を示します。

表 *8. NUM_IMAGES* フィールドの設定

NUM_IMAGES (4 バイト**)**		
状況 (2 バイト)	バージョン (2 バイト)	名前 (8 バイト)
状況 (2 バイト)	バージョン (2 バイト)	名前 (8 バイト)

表 *8. NUM_IMAGES* フィールドの設定 *(*続き*)*

NUM_IMAGES (4 バイト**)**		
状況 (2 バイト)	バージョン (2 バイト)	名前 (8 バイト)
状況 (2 バイト)	バージョン (2 バイト)	名前 (8 バイト)
状況 (2 バイト)	バージョン (2 バイト)	名前 (8 バイト)
状況 (2 バイト)	バージョン (2 バイト)	名前 (8 バイト)
状況 (2 バイト)	バージョン (2 バイト)	名前 (8 バイト)
状況 (2 バイト)	バージョン (2 バイト)	名前 (8 バイト)

NUM_IMAGES フィールドは、合計 BUF フィールドに含まれている TCP_IMAGE の得エントリー数を示します。返された NUM_IMAGES が 0 の場合、TCP/IP イメージは存在しません。

状況フィールドは次の情報で構成されます。

状況フィールド
: 意味

X'8xxx'
: アクティブ

X'4xxx'
: 終了中

X'2xxx'
: ダウン

X'1xxx'
: 停止または停止中

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO フィールドは無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
>=0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

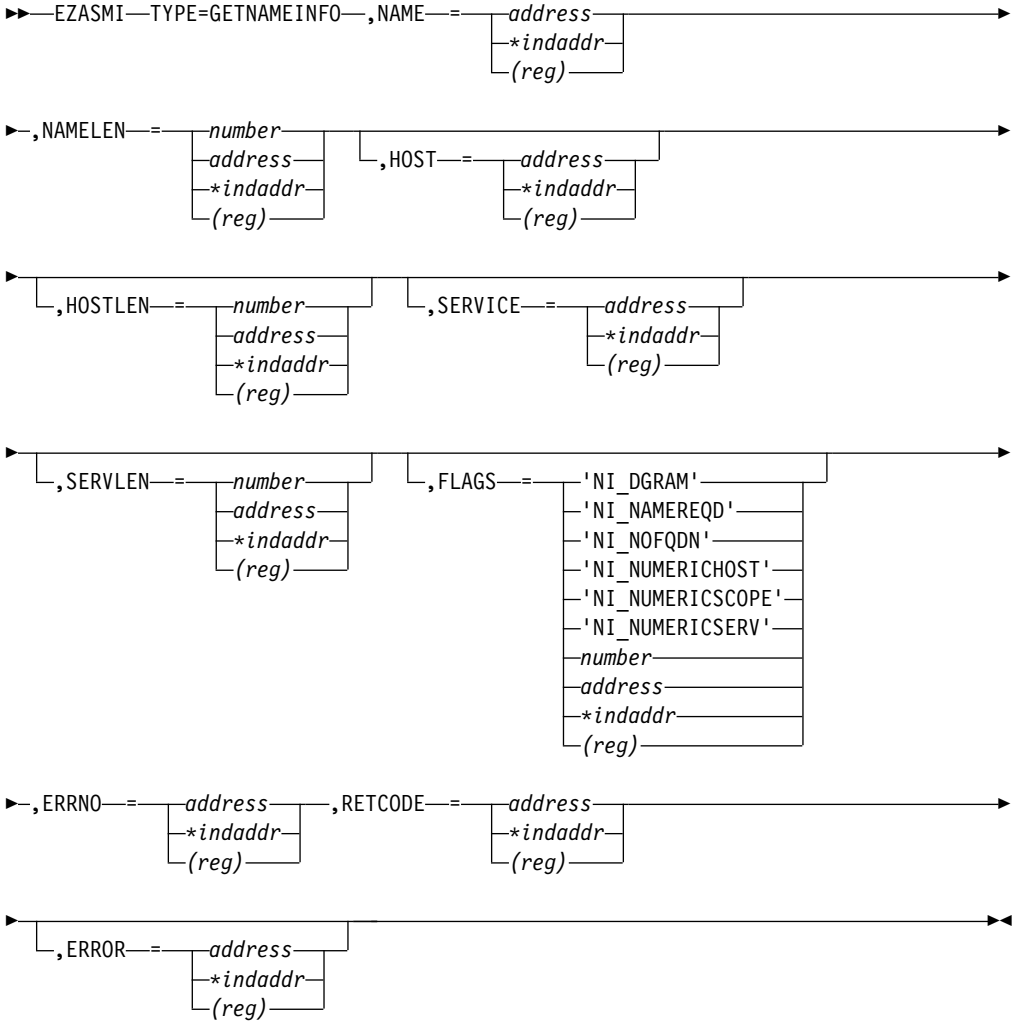
GETNAMEINFO

GETNAMEINFO マクロは、マクロで指定されたソケット・アドレスのノード名およびサービス・ロケーションを返します。

正常に完了した場合、GETNAMEINFO は、要求されていれば、指定したバッファーで命名されたノードおよびサービスを返します。

重要: この関数呼び出しは TCP/IP for z/VSE では使用できません。

フォーマット



パラメーター

NAME

変換される IPv4 または IPv6 ソケット・アドレス構造体。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。

AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。IPv4 ソケット・アドレス構造体は、以下のフィールドを指定する必要があります。

FAMILY

ハーフワード 2 進数であり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT

ハーフワード 2 進数であり、ポート番号を指定します。

IPv4-ADDRESS

フルワード 2 進数であり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED

8 バイトの予約フィールドです。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを指定する必要があります。

NAMELEN

IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY

IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT

ハーフワード 2 進数であり、ポート番号を指定します。

FLOW-INFO

このフィールドは TYPE=GETNAMEINFO マクロにより無視されます。

IPv6-ADDRESS

16 バイト 2 進数フィールドであり、128 ビットの IPv6 インターネット・アドレスをネットワーク・バイト・オーダーで指定します。

SCOPE-ID

フルワード 2 進数フィールドであり、インターフェース・インデックスとしての IPv6 アドレスの有効範囲を指定します。

IPv6-ADDRESS のアドレスがリンク・ローカル・アドレスであり、さらに HOST パラメーターも指定されているのでない限り、リゾルバーは SCOPE_ID フィールドを無視します。

NAMELEN

入力パラメーター。フルワード 2 進数フィールド。NAME 引数により示されるソケット・アドレス構造体の長さ。

HOST

入力では、返された解決済みホスト名を保持できるストレージであり、入力ソケット・アドレスに対して最大で 255 バイトの長さにすることができます。解決済みホスト名を入れるために不適切なストレージが指定された場合、リゾルバーは指定されたストレージまでのホスト名を返し、切り捨てが行われる場合もあります。ホスト名が見つからない場合、その名前の代わりに数値形式のホスト・アドレスが返されます。ただし、NI_NAMEREQD オプションが指定されているが、ホスト名が見つからない場合は、エラーが返されます。以下のパラメーターのグループのうち、1 つか両方が必要です。

- HOST および HOSTLEN パラメーター
- SERVICE および SERVLEN パラメーター

そうでない場合には、エラーが発生します。照会される HOST 名は、HOSTLEN まで、または最初の 2 進 0 までで構成されます。

IPv6-ADDRESS 値がリンク・ローカル・アドレスであり、SCOPE_ID インターフェース・インデックスが非ゼロである場合、スコープ情報が、*host%scope information* 形式を使用して解決済みホスト名に追加されます。スコープ情報は、数値形式の SCOPE_ID インターフェース・インデックス、または SCOPE_ID インターフェース・インデックスに関連付けられたインターフェース名にすることができます。返される形式を選択するには、NI_NUMERICSCOPE オプションを使用します。結合されたホスト名とスコープ情報は、255 バイト以下です。

HOSTLEN

フルワード 2 進数フィールドであり、返された解決済みホスト名を入れるために使用されるホスト・ストレージの長さが入ります。入力で HOSTLEN が 0 の場合、解決済みホスト名は返されません。HOSTLEN 値は、返される最長のホスト名、またはホスト名とスコープ情報の組み合わせの長さ以上でなければなりません。TYPE=GETNAMEINFO は、ホスト名、またはホスト名とスコープ情報の組み合わせを、HOSTLEN 値により指定された長さまで返します。出力では、HOSTLEN には、返されるホスト名、またはホスト名とスコープ情報の組み合わせの長さが入ります。これはオプション・フィールドですが、このフィールドを指定する場合は、HOST 値をコーディングする必要もあります。以下のパラメーターのグループのうち、1 つか両方が必要です。

- HOST および HOSTLEN パラメーター
- SERVICE および SERVLEN パラメーター

そうでない場合には、エラーが発生します。

SERVICE

入力では、返された解決済みサービス名を保持できるストレージであり、入力ソケット・アドレスに対して最大で 32 バイトの長さにすることができます。解決済みサービス名を入れるために不適切なストレージが指定された場合、リゾルバーは指定されたストレージまでサービス名を返します。さらに、切り捨てが行われる場合もあります。サービス名が見つからない場合、または NI_NUMERICSERV が FLAGS オペランドに指定されている場合、その名前の代わりに表示形式のサービス・アドレスが返されます。これはオプション・フィールドですが、このフィールドを指定する場合は、SERVLEN パラメーターをコーディングする必要もあります。照会される SERVICE 名は、SERVLEN まで、または最初の 2 進ゼロまでで構成されます。以下のパラメーターのグループのうち、1 つか両方が必要です。

- HOST および HOSTLEN パラメーター
- SERVICE および SERVLEN パラメーター

そうでない場合には、エラーが発生します。

SERVLEN

最初は入力パラメーター。フルワード 2 進数フィールドであり、返された解決済みサービス名を入れるために使用される SERVICE ストレージの長さが入ります。入力で SERVLEN が 0 の場合、サービス名情報は返されません。SERVLEN は、返される最長のサービス名の長さ以上でなければなりません。TYPE=GETNAMEINFO は、SERVLEN により指定された長さまでサービス名を返します。出力では、SERVLEN には、返される解決済みサービス名の長さが入ります。これはオプション・フィールドですが、これを指定する場合は、SERVICE パラメーターもコーディングする必要があります。以下のパラメーターのグループのうち、1 つか両方が必要です。

- HOST および HOSTLEN パラメーター
- SERVICE および SERVLEN パラメーター

そうでない場合には、エラーが発生します。

FLAGS

フルワード 2 進数フィールド。これはオプションのフィールドです。FLAGS 引数は、以下のリテラル値またはフルワード 2 進数フィールド値にすることができます。

リテラル値	バイナリー値	**10** 進数値	説明
'NI_NOFQDN'	X'00000001'	1	完全修飾ドメイン・ネームの NAME 部分を返します。
'NI_NUMERICHOST'	X'00000002'	2	数値形式のホスト・アドレスのみを返します。
'NI_NAMEREQD'	X'00000004'	4	ホスト名が見つからない場合にエラーを返します。
'NI_NUMERICSERV'	X'00000008'	8	数値形式のサービス・アドレスのみを返します。

リテラル値	バイナリー値	**10** 進数値	説明
'NI_DGRAM'	X'00000010'	16	サービスがデータグラム・サービスであることを示します。デフォルトの動作では、サービスをストリーム・サービスと想定します。
'NI_NUMERICSCOPE'	X'00000020'	32	該当する場合、数値形式の SCOPE-ID インターフェース・インデックスのみを返します。

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO フィールドは無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ERROR

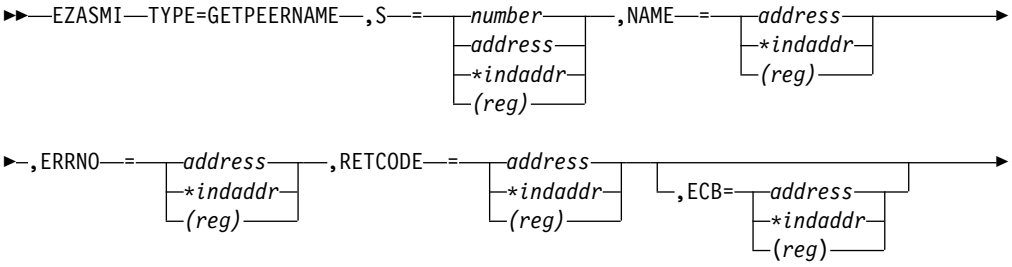
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

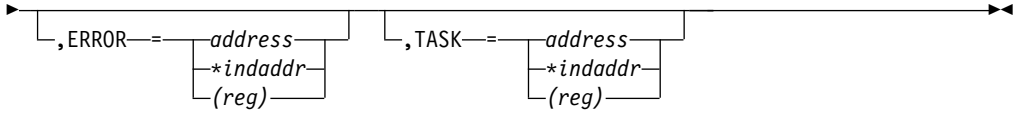
GETPEERNAME

GETPEERNAME マクロは、ローカル・ソケットの接続先であるリモート・ソケットの名前を戻します。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

フォーマット





パラメーター

S ハーフワード 2 進数のアドレスまたは値であり、アドレスを求めるリモート・ピアに接続されたローカル・ソケットを指定します。

NAME

最初は、ピア名構造体を指します。これには呼び出し完了時に、S で示されたローカル・ソケットに接続されたリモート・ソケットの IPv4 または IPv6 アドレス構造体が入ります。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

FAMILY

ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT

ハーフワード 2 進数フィールドであり、クライアントのポート番号を指定します。

IPv4-ADDRESS

フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED

2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

NAMELEN

IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY

IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT

ハーフワード 2 進数フィールドであり、クライアントのポート番号に設定されます。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の値の場合、このフィールドの中にはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールド。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

GETSOCKNAME

GETSOCKNAME マクロは、ソケット名を NAME で示された構造体に保管し、バインドされたソケットへのアドレスを戻します。

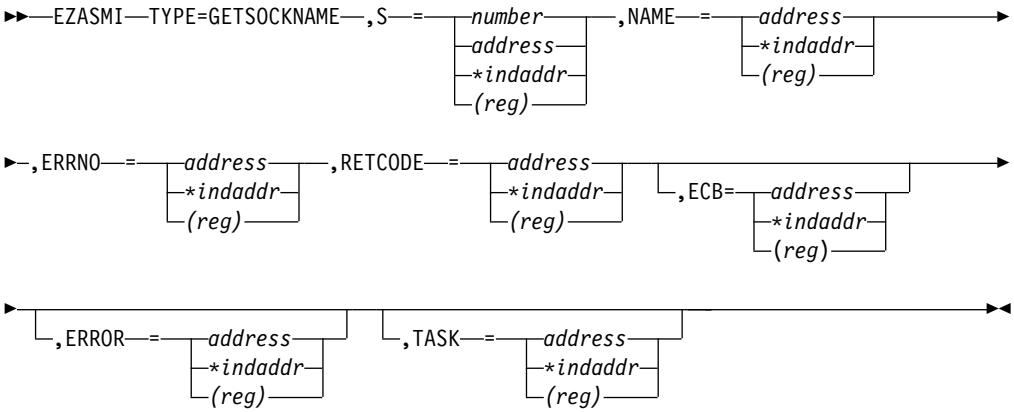
ソケットがバインドされているアドレスがない場合、このマクロは、FAMILY フィールドを設定し、構造体の残りの部分はゼロに設定して戻ります。

ストリーム・ソケットには、BIND、CONNECT、または ACCEPT への呼び出しが成功するまでは、名前は割り当てられません。

ソケットがポートに暗黙的にバインドされた後、そのソケットに割り当てられたポートを判別するために、GETSOCKNAME マクロを使用できます。アプリケーションが、前もって BIND を呼び出さずに CONNECT を呼び出した場合、CONNECT マクロは、ソケットにポートを割り当てることによって、必要なバインドを完了します。ソケットに割り当てられたポートを GETSOCKNAME を発行することによって判別できます。

重要: IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

フォーマット



パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、ソケット記述子を指定します。

NAME

最初は、アプリケーションが IPv4 または IPv6 ソケット・アドレス構造体へのポインターを設定し、呼び出しの完了時に、その構造体にソケット名が入ります。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

FAMILY

ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、このソケットにバインドされたポート番号を指定します。ソケットがバインドされていない場合、ゼロが戻されます。

IPv4-ADDRESS
フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

NAMELEN
IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、このソケットにバインドされたポート番号を指定します。ソケットがバインドされていない場合、ゼロが戻されます。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の値の場合、このフィールドの中にはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

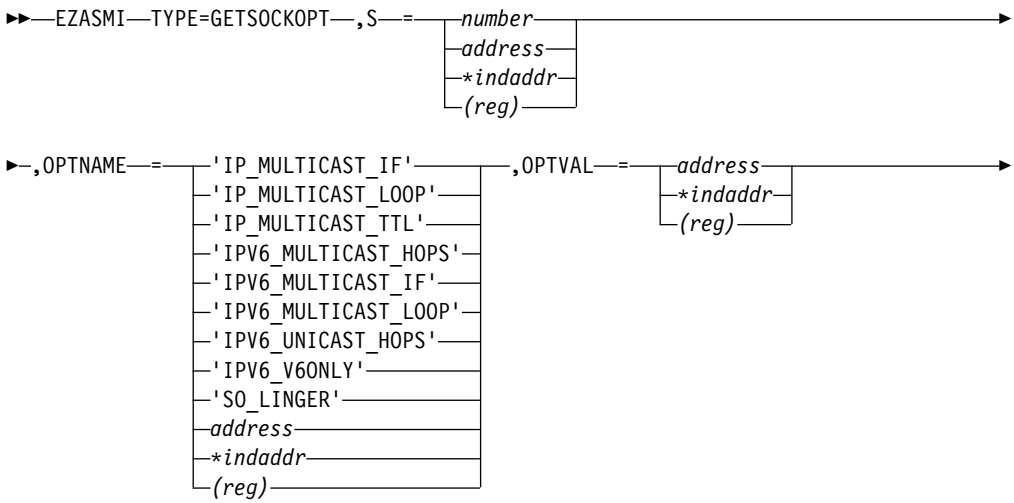
ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

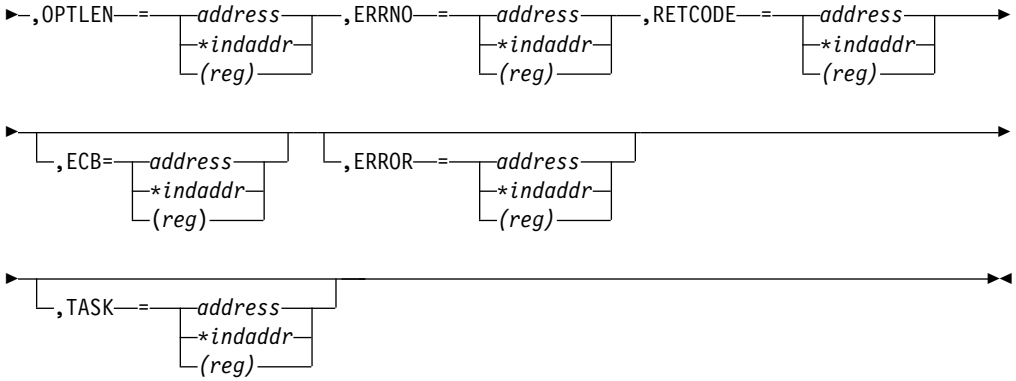
TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

GETSOCKOPT

GETSOCKOPT マクロは、 425 ページの『SETSOCKOPT』 マクロを使用して設定された、ソケットに関連付けられたオプションを取得します。各ソケットのオプションは、次のパラメーターで表します。GETSOCKOPT マクロを発行する場合は、取得するオプションを指定する必要があります。

フォーマット





パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、オプションを取得したいソケットのソケット記述子を指定します。

OPTNAME
入力パラメーター。OPTNAME は、GETSOCKOPT を発行する前に、以下のいずれかのオプションに設定してください。

IP_MULTICAST_IF
このオプションは、ソケット・アプリケーションからアウトバウンド・マルチキャスト・データグラムを送信する際に使用する IPv4 インターフェース・アドレスを取得するために使用します。これは IPv4 専用のソケット・オプションです。

注**:** マルチキャスト・データグラムを送信できるインターフェースは、一度に 1 つのみです。

IP_MULTICAST_LOOP
送信ホスト自体が属するグループに送信されたマルチキャスト・データグラムについて、マルチキャスト・データグラムのコピーをループバックするかどうかを指定するには、このオプションを使用します。デフォルトでは、データグラムをループバックします。これは IPv4 専用のソケット・オプションです。

IP_MULTICAST_TTL
このオプションを使用して、発信マルチキャスト・データグラムの IP データグラム存続時間 (ホップ) を取得します。デフォルト値は '01'x です。これは、マルチキャストがローカル・サブネットでのみ使用可能であることを意味します。これは IPv4 専用のソケット・オプションです。

IPV6_MULTICAST_HOPS
このオプションを使用して、発信マルチキャスト・パケットに使用するホップ限界を取得します。これは IPv6 専用のソケット・オプションです。

IPV6_MULTICAST_IF
このオプションを使用して、ソケット・アプリケーションからアウ

トバウンド・マルチキャスト・データグラムを送信する際に使用する IPv6 インターフェースの索引を取得します。これは IPv6 専用のソケット・オプションです。

IPV6_MULTICAST_LOOP
送信ホスト自体が属するグループに対してデータグラムを送信する場合、ローカル配信のため、IP レイヤーによって発信インターフェース上でマルチキャスト・データグラムをループバックするかどうかを指定するには、このオプションを使用します。デフォルトでは、マルチキャスト・データグラムをループバックします。これは IPv6 専用のソケット・オプションです。

IPV6_UNICAST_HOPS
このオプションを使用して、発信ユニキャスト IPv6 パケットに使用するホップ限界を取得します。これは IPv6 専用のソケット・オプションです。

IPV6_V6ONLY
このオプションを使用して、ソケットを IPv6 パケットのみの送受信に制限するかどうかを指定します。デフォルトでは、IPv6 パケットのみの送受信に制限しません。これは IPv6 専用のソケット・オプションです。

SO_LINGER
SO_LINGER の状況を要求します。

- SO_LINGER オプションが使用可能にされていて、データ伝送が完了していない場合、CLOSE マクロは、データが送信されるまで、または接続がタイムアウトになるまで、呼び出し側プログラムをブロックします。
- SO_LINGER が使用可能にされていない場合、CLOSE 呼び出しは、呼び出し側プログラムをブロックせずに戻り、TCP/IP はデータを送信関数を続行します。通常、データ送信関数は完了してデータは送信されますが、送信が完了する前に TCP/IP がタイムアウトになる可能性があるので、それが保証されるわけではありません。

OPTVAL
出力パラメーター。

- OPTNAME に SO_LINGER が指定されている場合、次の構造体が戻されます。

```
ONOFF       DS   F
LINGER      DS   F
```

  – ONOFF にゼロ以外の値が戻された場合、このオプションが使用可能になっていることを示します。ゼロ値は、使用不可になっていることを示します。
  – LINGER 値は、CLOSE 呼び出しが発行された後に TCP/IP がデータ送信を続行する時間 (秒) を示します。LINGER 時間の設定方法については、 311 ページの『SETSOCKOPT』を参照してください。

OPTLEN
入力パラメーター。フルワード 2 進数フィールドであり、OPTVAL に戻されるデータの長さを含みます。

- SO_LINGER の場合、OPTVAL は 2 個のフルワードを含み、OPTLEN は 8 (2 個のフルワード) に設定されます。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の値の場合、このフィールドの中にはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注: このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

GIVESOCKET

GIVESOCKET マクロは、別のプログラムで発行される TAKESOCKET マクロのためにソケットを使用可能にします。

GIVESOCKET マクロは、任意の接続済みストリーム・ソケットを指定することができます。一般的には、GIVESOCKET マクロは、サブタスクに渡すソケットを作成する並行サーバー・プログラムが発行します。

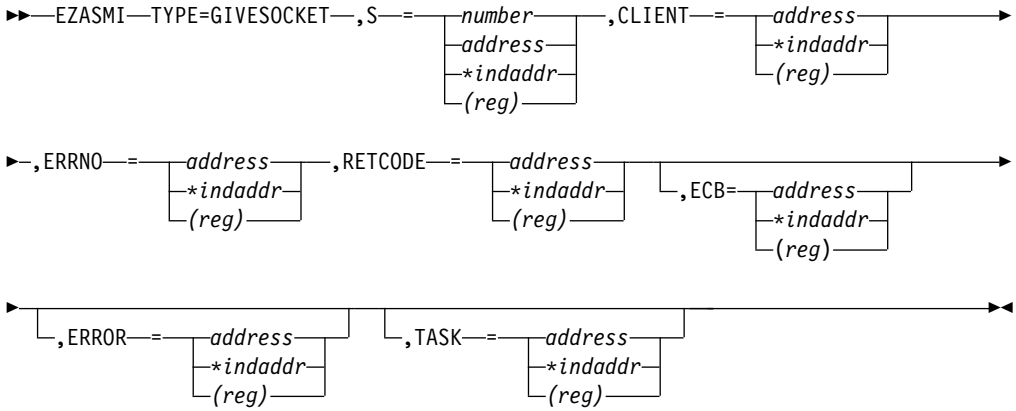
あるソケットに対して GIVESOCKET マクロを発行したプログラムでは、その後は同じソケットに対して CLOSE マクロのみを発行できます。

注: 並行サーバーと反復サーバーの両方が、このインターフェースを使用します。反復サーバーは、一時点では 1 つのクライアントを扱います。並行サーバーは複数のクライアントから接続要求を受信し、それらのクライアント要求を処理するサブタスクを作成します。サブタスクが作成されると、並行サーバーは新規ソケットを取

得し、その新規ソケットをサブタスクに渡し、自身をその接続から切り離します。CICS リスナー・プログラムは、並行サーバーの 1 例です。

重要: IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

フォーマット



パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、与えられるソケットの記述子を指定します。

CLIENT
ソケットが与えられる先のアプリケーションの ID を含む構造体です。

DOMAIN
入力パラメーター。フルワード 2 進数であり、クライアントのドメインを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2、または AF_INET6 を示す 10 進数の 19 に設定されます。

注: GIVESOCKET で指定されたソケットは、同じ DOMAIN、アドレス・ファミリー (AF_INET または AF_INET6) の TAKESOCKET によってのみ取得できます。

NAME
8 文字のフィールドであり、左寄せで右にはブランクを入れます。ソケットを取得するアプリケーションのアドレス・スペース名 (パーティション ID) に設定されます。このフィールドがブランクのままである場合、任意の z/VSE パーティションがソケットを取得できます。

TASK
8 文字のフィールドを指定し、ブランクに設定するか、またはソケットを取得する VSE サブタスクの ID に設定できます。このフィールドがブランクに設定されている場合、NAME フィールドに指定されたパーティション内の任意のサブタスクがソケットを取得できます。

RESERVED
20 バイトの予約フィールドです。このフィールドは必須ですが、内部的にのみ使用されます。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注: このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

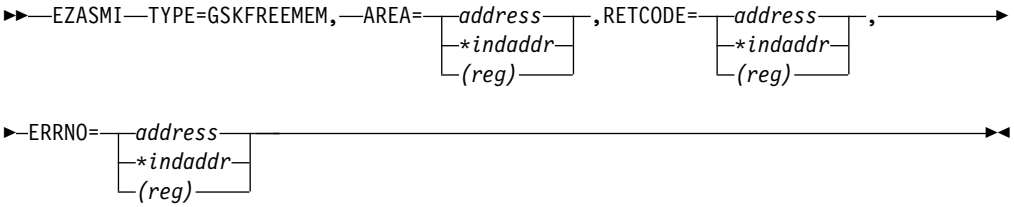
ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

GSKFREEMEM

この関数は、前の SSL 関数呼び出しでアプリケーションに渡されたメモリーを解放します。

フォーマット



パラメーター

AREA
入力パラメーター。前の SSL 呼び出しからアプリケーションに戻された、解放するメモリーのアドレスを指定します。

RETCODE
出力パラメーター。値 0 は、関数の正常終了を示します。RETCODE が負の値の場合、エラーが発生したことを示します。

ERRNO
出力パラメーター。詳しいエラー情報を示します。

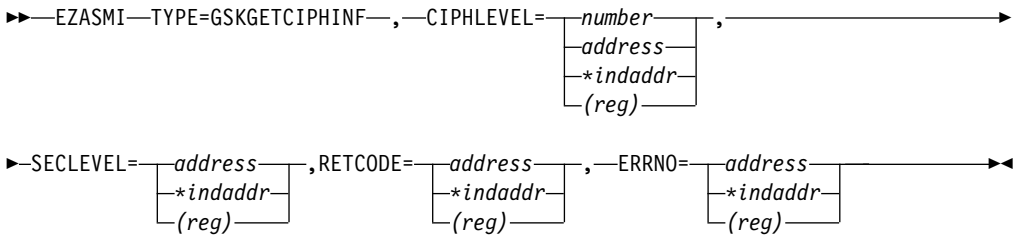
注: GSKGETDNBYLAB 関数によってヌル終了ストリングで返された識別名は、GSKFREEMEM を使用して解放する必要があります。

GSKGETCIPHINF

この関数は、SSL for VSE の暗号関連情報を要求します。

この情報は、システムがサポートできる暗号化レベルを判別し、SSL が使用できる暗号仕様のリストを戻します。これにより、アプリケーションはその実行時に、インストール済みアプリケーションが要求できる SSL 暗号化のレベルを判別することができます。

フォーマット



パラメーター

CIPHLEVEL
入力パラメーター。フルワード 2 進数のアドレスまたは値であり、戻される暗号情報のタイプを決定します。有効値は、次のものです。

1 米国/カナダ外で使用可能な暗号情報のみが戻されます (GSK_LOW_SECURITY)。

2 米国/カナダ外で使用可能な暗号情報と米国/カナダ国内用の暗号情報が戻されます (GSK_HIGH_SECURITY)。

SECLEVEL
出力パラメーター。104 バイトの領域 (アプリケーションによって割り振られます) を示し、ここにシステムが以下の情報を戻します。

4 バイト
システム SSL バージョン (常に GSK_VERSION3 を示す 3)

64 バイト
文字ストリング (x00 で終わる) であり、システムで使用が許可される SSL バージョン 3 暗号仕様を含みます (これらの仕様は GSKSSOCINIT 呼び出しの V3CIPHER パラメーターに渡すことができます)。

32 バイト
SSL for VSE は SSL バージョン 2 暗号仕様をサポートしないため、このフィールドには常に 2 進ゼロが埋められます。

4 バイト
以下のいずれかです。

1 GSK_SEC_LEVEL_US

2 GSK_SEC_LEVEL_EXPORT

3 GSK_SEC_LEVEL_EXPORT_FR

RETCODE
出力パラメーター。値 0 は、関数の正常終了を示します。RETCODE が 0 以外の値の場合は、エラーが発生したことを示します。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

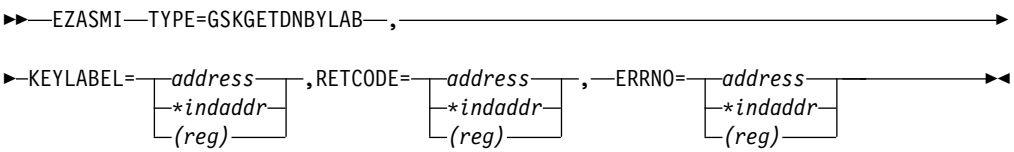
ERRNO
出力パラメーター。詳しいエラー情報を示します。

GSKGETDNBYLAB

この関数は、ある鍵の完全な識別名を、キー・データベース・ファイル内のその鍵のラベルに基づいて戻します。

この値は、GSKSSOCINIT 呼び出しの DNAME フィールドに使用できます。

フォーマット



パラメーター

KEYLABEL
入力パラメーター。キー・データベース・ファイル内の鍵ラベルが入っている文字ストリングを指します。ストリングは x00 で終わらなければなりません。

RETCODE
出力パラメーター。0 より大きい値は、関数の正常終了を示し、識別名の入った文字ストリングへのポインターを表します。値 0 はエラーを示します。

ERRNO
出力パラメーター。詳しいエラー情報を示します。

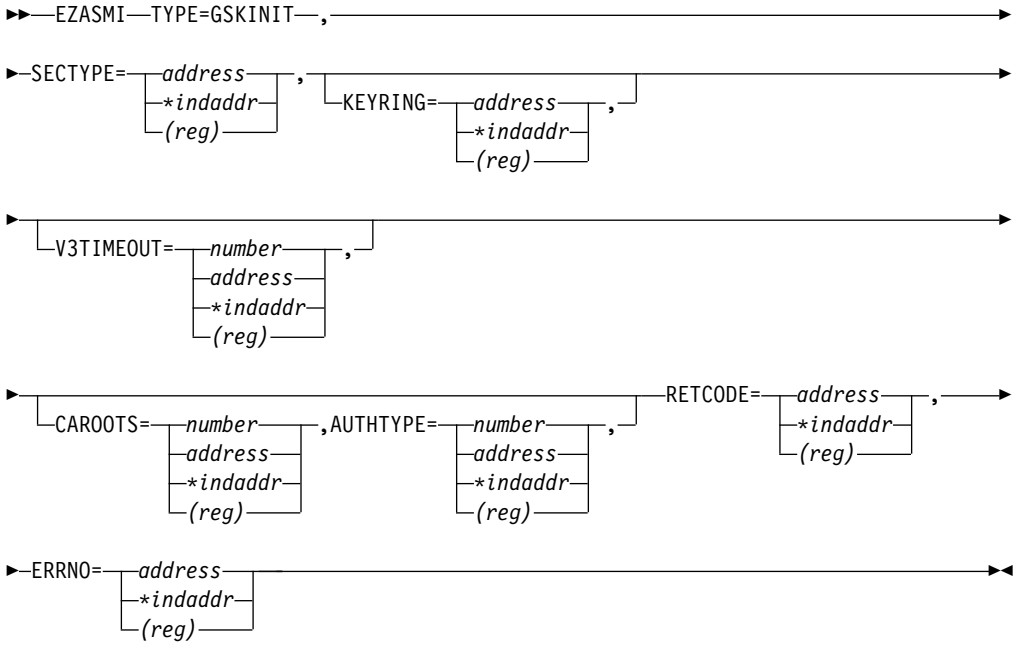
注: ヌル終了ストリングで返された識別名は、GSKFREEMEM 関数を使用して解放する必要があります。

GSKINIT

この関数は、現行パーティションについて、全般的な SSL for VSE 環境を設定します。

この関数が正常に完了した後、アプリケーションは、SSL for VSE インターフェースの呼び出し、およびセキュア・ソケット接続の作成と使用を行うことができます。

フォーマット



パラメーター

SECTYPE
入力パラメーター。受け入れ可能な最小限のセキュリティー・プロトコルを示す文字ストリングを指します。値は大文字で入力し、x00 で終わらせなければなりません。有効な値は次のとおりです (二重引用符は除きます)。

- SSL バージョン 3.0 を表す "SSL30"
- TLS バージョン 1.0 を表す "TLS31" (クライアント・アプリケーションにはサポートされません)

KEYRING
(オプション) 入力パラメーター。秘密鍵および証明書が保管されている "lib.sublib" を指定する文字ストリングを指します。ストリングは、x00 で終わらなければなりません。このパラメーターが使用される場合、後で GSKGETDNBYLAB 呼び出しを使用して、GSKSSOCINIT 呼び出しの

DNAME パラメーターに指定されるライブラリー・メンバー名を識別する必要があります。このパラメーターが指定されない場合、プロシージャー $SSL4VSE.PROC 内に定義されたデフォルトの "SSL for VSE" ファイルが使用されます (詳しくは、マニュアル「*TCP/IP for VSE Optional Features*」を参照してください)。

V3TIMEOUT
(オプション) 入力パラメーター。フルワード 2 進数のアドレスまたは値であり、SSL V3 セッション ID の有効期限を表す秒数を指定します。有効な範囲は 0 から 86400 秒 (1 日) です。このパラメーターが指定されていない場合、デフォルト値の 86400 が想定されます。

CAROOTS
(オプション) 入力パラメーター。フルワード 2 進数のアドレスまたは値であり、証明書の検証に使用する CA (認証局) ルートを指定します。以下の値がサポートされます。

0 ローカル・キー・データベース・ファイルからの CA ルートを使用して、証明書を検証します。

1 VSE と同じ認証局によって発行された証明書でのクライアント認証を許可します。

このパラメーターが指定されていない場合、デフォルト値の 0 が想定されます。

AUTHTYPE
(オプション) 入力パラメーター。フルワード 2 進数のアドレスまたは値であり、クライアント証明書の検証に使用する方式を指定します。CAROOTS フィールドが 1 に設定されている場合、このフィールドは必須であり、CAROOTS が 0 に設定されている場合は無視されます。サポートされる値は次のとおりです。

0 クライアントの証明書は、ローカル・キー・データベース・ファイルを使用して検証されます。

1 値 0 と同じ意味です。

2 値 0 と同じ意味です。

3 クライアントの証明書は検証されません。

RETCODE
出力パラメーター。値 0 は、関数の正常終了を示します。RETCODE が 0 以外の値の場合は、エラーが発生したことを示します。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

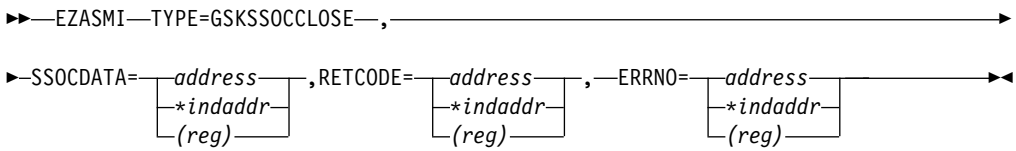
ERRNO
出力パラメーター。詳しいエラー情報を示します。

注**:** 後続の GSKINIT の呼び出しで、間に対応する GSKUNINIT 呼び出しがないものは、リジェクトされます。

GSKSSOCCLOSE

この関数は、セキュア・ソケット接続を終了し、その接続用のすべての SSL for VSE リソースを解放します。

フォーマット



パラメーター

SSOCDATA
入力パラメーター。EZASMI TYPE=GSKSSOCINIT によって RETCODE に戻される、GSKSOCDATA へのポインターです。

RETCODE
出力パラメーター。値 0 は、関数の正常終了を示します。RETCODE が 0 以外の値の場合は、エラーが発生したことを示します。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

ERRNO
出力パラメーター。詳しいエラー情報を示します。

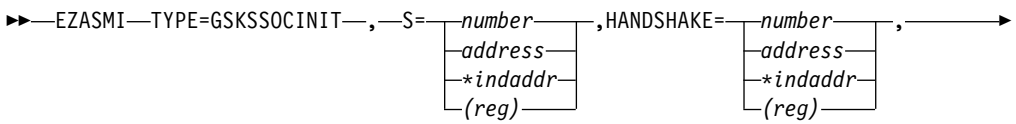
GSKSSOCINIT

この関数は、SSL for VSE がセキュア・ソケット接続の開始または受け入れを行うのに必要なデータ域を初期設定します。

この関数が正常に完了すると、セキュア・ソケット制御ブロック (これ以降、GSKSOCDATA で表します) へのポインターがアプリケーションに戻されます。このセキュア・ソケット接続を使用する他の呼び出しは、このポインターを使用する必要があります。

呼び出し中に、GSKSSOCINIT 呼び出しに指定された入力に基づいて、完全なハンドシェークが実行されます。SSL ハンドシェークの手順は SSL for VSE によって実行されますが、SSL ハンドシェーク中の SSL データの転送には、それに続くすべての読み取り/書き込み操作と同様、「通常の」RECV ルーチンと SEND ルーチン (EZAAPI 処理環境によって提供されるか、または、SKREAD および SKWRITE パラメーターを使用してアプリケーションによって提供されます) が使用されます。

フォーマット



パラメーター

S
入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、セキュア・ソケット接続用に初期化されるソケットのソケット記述子を指定します。

HANDSHAKE
入力パラメーター。フルワード 2 進数のアドレスまたは値であり、ハンドシェークがどのように実行されるのかを指定します。

0
クライアントとして SSL ハンドシェークを実行します (GSK_AS_CLIENT)。

1
サーバーとして SSL ハンドシェークを実行します (GSK_AS_SERVER)。

2
クライアント認証を必要とするサーバーとして SSL ハンドシェークを実行します (GSK_AS_SERVER_WITH_CLIENT_AUTH)。

3
認証なしのクライアントとして SSL ハンドシェークを実行します (GSK_AS_CLIENT_NO_AUTH)。

DNAME
オプションの入力パラメーターです。キー・データベース・ファイル内の使用したい項目 (証明書) の識別名またはラベルを表す文字ストリングを指します。この文字ストリングは、x00 で終わらなければなりません。デフォルトのキー・データベース・ファイル項目を使用するには、このパラメーターを省略します。キー・データベース・ファイル項目の識別名は、EZASMI GETDNBYLAB 関数呼び出しを介して判別できます。

SECTYPE
出力パラメーター。最小限の受け入れ可能なセキュリティー・プロトコルを

示す文字ストリングのアドレスが保管されているフルワードをポイントします。この文字ストリングは、x00 で終わります。有効な値は次のとおりです (二重引用符は除きます)。

- SSL バージョン 3.0 を表す "SSL30"
- TLS バージョン 1.0 を表す "TLS31"

V3CIPHER

オプションの入力パラメーターです。 SSL 暗号のリストが入っている文字ストリングです。このストリングは、 518 ページの『z/OS SSL API』 gsk_get_cipher_info の説明で示されているとおり、16 進値で構成されます。

V3CIPHSEL

出力パラメーター。2 バイトの領域 (アプリケーションによって用意される) を指します。この領域には、このセッション用に選択された SSL バージョン 3.0 暗号仕様に対応する値 (例: x0009) が保管されます。

CERTINFO

オプションの出力パラメーターです。クライアントの証明書からの識別名コンポーネントのアドレスが保管されるフルワードを指します。このパラメーターが有効なのは、SSL を使用するサーバーに対してクライアント認証が要求されている場合だけです。この領域のレイアウトは、次のとおりです。

4 バイト
: Base64 証明書本文へのポインター

4 バイト
: Base64 証明書本文の長さ

4 バイト
: この接続のセッション ID へのポインター

4 バイト
: 新しいセッションであるかどうかを示すフラグ

4 バイト
: 証明書シリアル番号へのポインター

4 バイト
: クライアントの共通名へのポインター

4 バイト
: 市町村へのポインター

4 バイト
: 都道府県へのポインター

4 バイト
: 国へのポインター

4 バイト
: 組織へのポインター

4 バイト
: 組織部門へのポインター

4 バイト
: 発行者の共通名へのポインター

4 バイト
: 発行者の市町村へのポインター

4 バイト
: 発行者の都道府県へのポインター

4 バイト
発行者の国へのポインター
4 バイト
発行者の組織へのポインター
4 バイト
発行者の組織部門へのポインター

SKREAD
オプションの入力パラメーターです。アプリケーションが用意した、SSL for VSE 用のソケット読み取り関数を実行するルーチンを指します。このルーチンは、以下の要件を満たさなければなりません。

- 実際の読み取りには、EZASMI READ または RECV 呼び出しを使用しなければなりません。
- 先頭からのバイト (EZASMI TYPE=TASK,STORAGE=DSECT/CSECT マクロの TIECLEN 等価を使用) に、GSK 呼び出しで使用される TIE がコピーされた、独自の TIE (タスク・インターフェース・エレメント) を使用しなければなりません。

このパラメーターが指定されない場合、EZAAPI 処理環境によって提供される読み取りルーチンが使用されます。

例:

メインルーチン

```
=====================================================================
........ ......
         EZASMI TYPE=GSKSSOCINIT,     Issue GSKSSOCINIT call     x
               .....
               SKREAD=*SKREADA,   SKREAD routine                 x
               .....
SKREADA  DC    V(T9SKREAD)
MAINTIE  EZASMI TYPE=TASK,STORAGE=CSECT     Task Interface Element
         ENTRY MAINTIE
........ .......
```

サブルーチン (メインルーチンにリンクされる)

```
=====================================================================
T9SKREAD START X'00'
T9SKREAD AMODE ANY
T9SKREAD RMODE ANY
         STM   R14,R12,12(R13)    Save Caller's Registers
         LR    R3,R15             Change base register to R3
         USING T9SKREAD,R3
         LR    R11,R1             Save addr of parameter list
* ******************************************************************
* Allocate our working storage
* ******************************************************************
         LA    R0,T9SKRDYL        Load the length
         GETVIS ADDRESS=(1),LENGTH=(0)
         LTR   R15,R15            Test the return code
         BZ    T9SKR010           ok
         SLR   R15,R15            not ok
         BCTR  R15,0              Set bad return code
         B     T9SKR090           Back to caller
T9SKR010 ST    R13,4(,R1)         Save caller's reg 13
         ST    R1,8(,R13)         Save our save area address
         LR    R13,R1             Load our save area address
         USING T9SKRDYN,R13       Base or work area
* ******************************************************************
* Process request
* ******************************************************************
```

```
         L     R6,AMTIE           Load addr of main TIE
         MVC   T9SKRTIE(TIECLEN),0(R6)  Copy first 24 bytes
         L     R6,0(R11)          Get addr of socket descriptor
         MVC   T9RSOCK,0(R6)      Move to local field
         L     R6,4(R11)          Get addr of buffer
         ST    R6,T9RBUFA         Move to local field
         L     R6,8(R11)          Get length of buffer
         MVC   T9RBUFL,0(R6)      Move to local field
         EZASMI TYPE=READ,        Read request                      c
               S=T9RSOCK+2,       for this socket descriptor        c
               BUF=*T9RBUFA,      to this buffer                    c
               NBYTE=T9RBUFL,     with this length                  c
               TASK=T9SKRTIE,     own task storage                  c
               ERRNO=T9RERRNO,    own ERRNO                         c
               RETCODE=T9RRETCD   own RETCODE
         L     R15,T9RRETCD       Move RETCODE to reg 15
T9SKR090 L     R13,4(,R13)        Load caller's reg 13
         L     R14,12(R13)
         LM    R0,R12,20(R13)
         BR    R14                Back to caller
* --- Constants ----------------------------------------------------
AMTIE    DC    V(MAINTIE)         Address of main TIE
         EZASMI TYPE=TASK,STORAGE=DSECT   TIE DSECT
* --- Dynamic work area --------------------------------------------
T9SKRDYN DSECT                    Dynamic Storage for this module
T9SKRSAV DS    18F                Own savearea (MUST: begin of dyn)
T9SKRTIE DS    XL(TIELENTH)       Own TIE
T9RSOCK  DS    F                  Socket descriptor
T9RBUFA  DS    F                  Addr of read buffer
T9RBUFL  DS    F                  Length of read buffer
T9RERRNO DS    F                  Addr of errno value
T9RRETCD DS    F                  Addr of retcode value
T9SKRDYL EQU   *-T9SKRDYN         Length of dynamic storage
*------------------------------------------------------------------
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*        END   T9SKREAD
```

SKWRITE

オプションの入力パラメーターです。アプリケーションが用意した、SSL for VSE 用のソケット書き込み関数を実行するルーチンを指します。このルーチンは、以下の要件を満たさなければなりません。

- 実際の書き込みには、EZASMI WRITE または SEND 呼び出しを使用しなければなりません。
- 先頭からのバイト (EZASMI TYPE=TASK,STORAGE=DSECT/CSECT マクロの TIECLEN 等価を使用) に、GSK 呼び出しで使用される TIE がコピーされた、独自の TIE (タスク・インターフェース・エレメント) を使用しなければなりません。

このパラメーターが指定されない場合、EZAAPI 処理環境によって提供される「書き込み」ルーチンが使用されます。

例:

SKREAD 例に似ています。

REASCODE
出力パラメーター。GSKSSOCINIT 呼び出しの失敗の理由コードが保管されるフルワードを指します。値 0 は、関数の正常終了を示します。

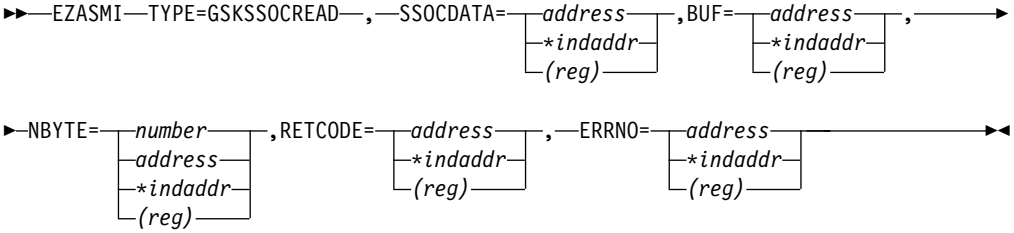
RETCODE
出力パラメーター。REASCODE が 0 の場合、RETCODE パラメーターには、GSKSOCDATA 構造体へのポインターが入っています。これは、後続の SSL for VSE 操作で使用される必要があります。

ERRNO
出力パラメーター。詳しいエラー情報を示します。

GSKSSOCREAD

この関数は、セキュア・ソケット接続でデータを受信します。

フォーマット



パラメーター

SSOCDATA
入力パラメーター。EZASMI TYPE=GSKSSOCINIT によって RETCODE に戻される、GSKSOCDATA のアドレスです。

BUF 入力パラメーター。ユーザーが用意する、データが保管されるバッファーのアドレスです。

NYBTE
入力パラメーター。フルワード 2 進数のアドレスまたは値であり、データ・バッファーの長さを指定します。データ・バッファーの長さは、64Kb か、または、受信する最大送信バッファーより少なくとも 32 バイトは大きい値でなければなりません。

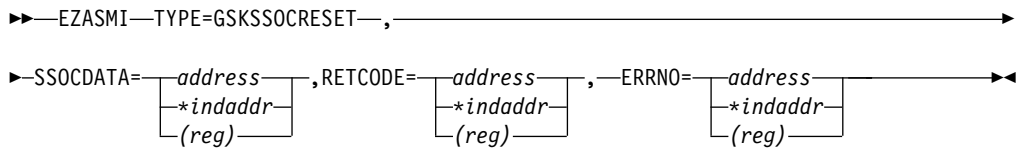
RETCODE
出力パラメーター。0 または 0 より大きい値は、関数の正常終了を示し、読み取られたバイト数を表します。RETCODE が負の値の場合、エラーが発生したことを示します。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

ERRNO
出力パラメーター。詳しいエラー情報を示します。非ブロッキング・ソケットの場合、受信したデータがない場合に、GSKSSOCREAD は EWOULDBLOCK に設定された ERRNO を戻すことがあります。

GSKSSOCRESET

この関数は、セッションのセキュリティー・パラメーター (暗号鍵など) をリフレッシュします。

フォーマット



パラメーター

SSOCDATA
入力パラメーター。EZASMI TYPE=GSKSSOCINIT によって RETCODE に戻される、GSKSOCDATA のアドレスです。

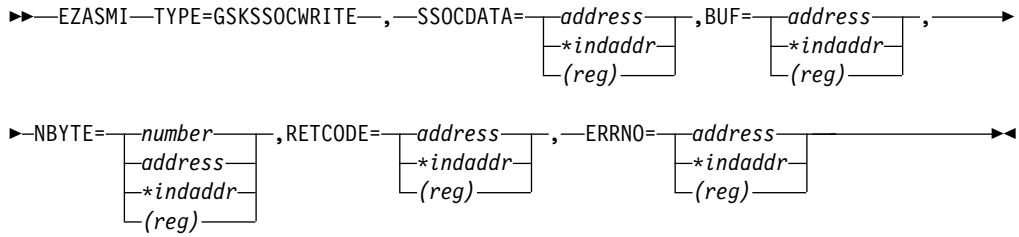
RETCODE
出力パラメーター。値 0 は、関数の正常終了を示します。RETCODE が負の値の場合、エラーが発生したことを示します。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

ERRNO
出力パラメーター。詳しいエラー情報を示します。

GSKSSOCWRITE

この関数は、セキュア・ソケット接続でデータを送信します。

フォーマット



パラメーター

SSOCDATA
入力パラメーター。EZASMI TYPE=GSKSSOCINIT によって RETCODE に戻される、GSKSOCDATA のアドレスです。

BUF 入力パラメーター。送信されるデータのアドレスです。

NYBTE
入力パラメーター。フルワード 2 進数のアドレスまたは値であり、送信されるバイト数を指定します。

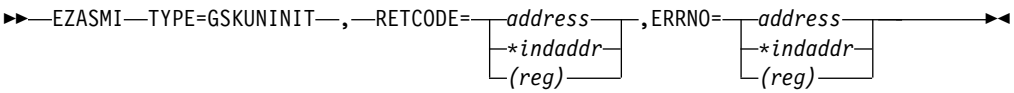
RETCODE
出力パラメーター。0 または 0 より大きい値は、関数の正常終了を示し、送信されたバイト数を表します。RETCODE が負の値の場合、エラーが発生したことを示します。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

ERRNO
出力パラメーター。詳しいエラー情報を示します。

GSKUNINIT

GSKUNINIT 呼び出しは、SSL 環境に関する現行の設定全体を除去します。セッションのタイムアウト値や SSL プロトコルなどのフィールドが除去されます。

フォーマット



パラメーター

RETCODE
出力パラメーター。値 0 は、関数の正常終了を示します。RETCODE が 0 以外の値の場合は、エラーが発生したことを示します。エラー戻りコードの詳しい説明については、VSE ライブラリー・メンバー SSLVSE.A または「*TCP/IP for VSE Optional Features*」を参照してください。

ERRNO
出力パラメーター。詳しいエラー情報を示します。

INITAPI

INITAPI マクロは、アプリケーションを TCP/IP インターフェースに接続します。

COBOL、PL/I、またはアセンブラー言語で書かれたほとんどすべてのソケット・プログラムは、他のソケット・マクロを発行する前に INITAPI マクロを発行する必要があります。

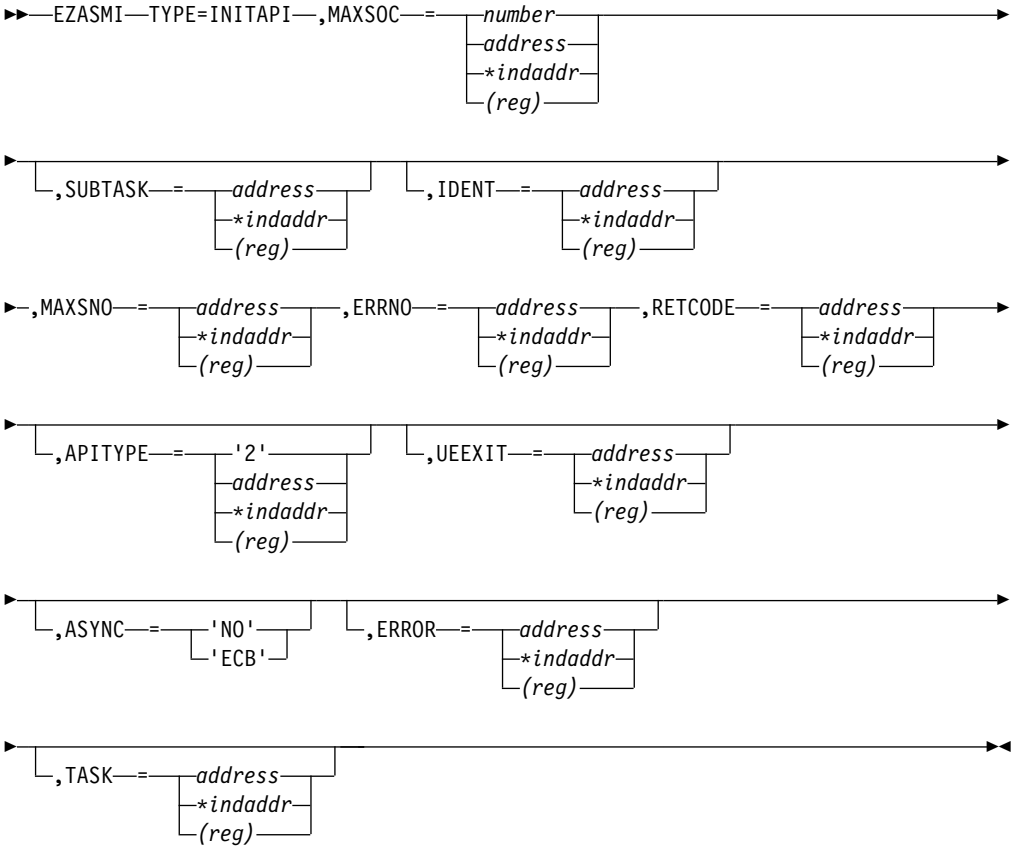
注**:** デフォルトの INITAPI であってもやはり TERMAPI が発行されることを必要とするため、常に INITAPI コマンドをコーディングすることをお勧めします。

以下の呼び出しが最初に発行される場合、デフォルトの INITAPI 呼び出しが作成されるため、これらの呼び出しはこのルールの例外です。

- GETCLIENTID
- GETHOSTID

- GETHOSTNAME
- SELECT
- SELECTEX
- SOCKET
- TAKESOCKET

フォーマット



パラメーター

MAXSOC

入力パラメーター。ハーフワード 2 進数フィールドであり、このアプリケーションに対してサポートされる最大ソケット数を指定します。現在、TCP/IP for z/VSE はこの入力を無視し、サポートされる最大ソケット数をデフォルトの 8192 に設定します。ソケット記述子の数値の範囲は、0 から 8191 です。

SUBTASK

8 バイトのフィールドであり、1 つのアドレス・スペース内の複数のサブタスクを識別するのに使用される固有のサブタスク ID を含みます。独自のジョブ名をサブタスク名の一部として使用してください。そうすれば、 同じアドレス・スペースから複数の INITAPI コマンドを発行しても、 それぞれの SUBTASK パラメーターは固有になります。これが指定されないか、8

個のブランクが指定された場合、デフォルトのサブタスク名が使用されます。バッチ環境では、次のようになります。

バイト **0-2**
ジョブ名の先頭 3 文字

バイト **3**
16 進 F0

バイト **4-7**
VSE タスク ID

CICS トランザクション環境では、次のようになります。

バイト **0-2**
CICS EIBTRNID (トランザクション ID)

バイト **3**
16 進 F1

バイト **4-7**
CICS EIBTASKN (タスク番号)

IDENT
TCP/IP アドレス・スペースの ID と呼び出し側プログラムのアドレス・スペースの ID が入る構造体。アドレス・スペースからの INITAPI 呼び出しに IDENT を指定します。

TCPNAME
z/VSE 4.2 以降、このパラメーターを、このアプリケーションで使用するローカル TCP/IP スタックの選択に使用できるようになりました。この 8 バイトのパラメーターは、「SOCKET*nn*」または単に「*nn*」と設定できます (左寄せまたは右寄せし、6 つのブランクを埋め込む)。値「*nn*」で、TCP/IP スタートアップ JCL の ID パラメーターで指定された、選択された TCP/IP スタックの ID を判別します。

ADSNAME
このパラメーターは、EZA 処理環境で使用される TCP/IP インターフェース・ルーチンの名前の指定に使用できます。ここに何も指定しないと、IBM 提供の TCP/IP インターフェース・ルーチン EZASOH99 が使用されます。この指定は、JCL ステートメント // SETPARM [SYSTEM,] EZA$PHA='routine-name' で上書きできることに注意してください。

| 注: LE/C ソケット API マルチプレクサーを使用して、アプリケーションで使用する TCP/IP および SSL の実装を選択する方法については、 95 ページの『第 10 章 使用する TCP/IP と SSL の実装を選択』を参照してください。

MAXSNO
出力パラメーター。フルワードの 2 進数フィールドであり、 このアプリケーションに割り当てることのできる最大の記述子値を含みます。現在、TCP/IP for z/VSE は常に 8191 を戻します。

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

APITYPE

オプションの入力パラメーターです。ハーフワード 2 進数フィールドであり、次のように APITYPE を指定します。

2 APITYPE 2。これはデフォルトです。非同期マクロ API プログラムが、ソケット記述子当たり 1 つだけの未解決ソケット呼び出しを持つことを許可します。APITYPE=2 プログラムは、非同期および同期の両方の呼び出しを使用できます。

UEEXIT

すべてのパラメーターは無視されます。

ASYNC

オプションの入力パラメーターです。以下のいずれかです。

- リテラル 'NO' は、非同期サポートがないことを示します。
- リテラル 'ECB' は、ECB を使用した非同期サポートが使用されることを示します。

ERROR

入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK

入力パラメーター。プログラム内のタスク・ストレージ域の位置。

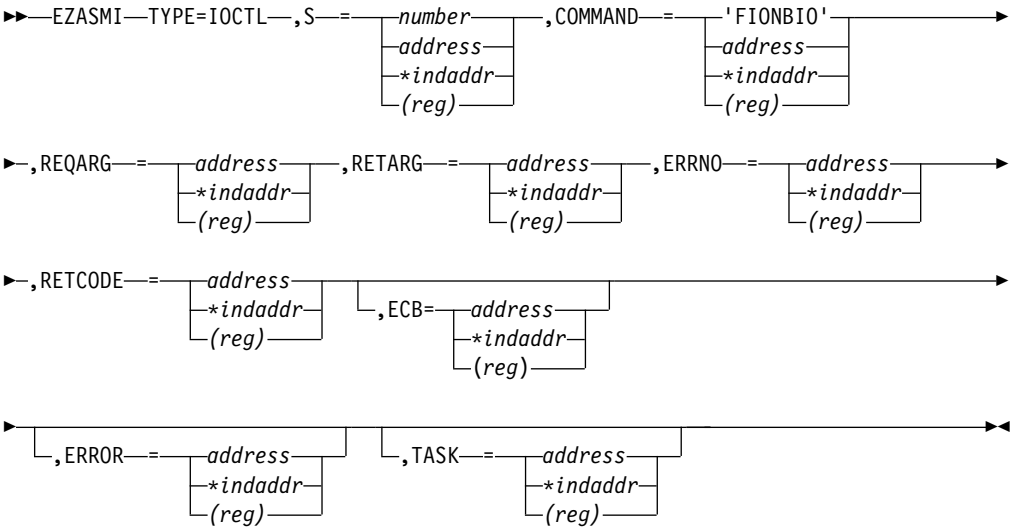
IOCTL

IOCTL マクロは、ソケットの特定の動作特性を制御することができます。

IOCTL マクロを発行する前に、制御する特性を表す値を COMMAND にロードしなければなりません。

注**:** IOCTL は、アドレス・ポインターがサポートされているプログラミング言語でのみ使用できます。

フォーマット



パラメーター

S　入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、制御されるソケットを指定します。

COMMAND

入力パラメーター。動作特性を制御するため、このフィールドは、以下のいずれかのシンボル名に設定されます。ビット・マスク中の値は、各シンボル名に関連しています。これらの名前のいずれか 1 つを指定すると、マスク中の 1 個のビットがオンになり、要求される動作特性を TCP/IP に知らせます。

'FIONBIO'

ブロッキング状況を設定またはクリアします。

REQARG および **RETARG**

呼び出し側プログラムと IOCTL との間で引き渡される引数を指します。引数の長さは、COMMAND 要求によって決まります。REQARG は入力パラメーターであり、引数を IOCTL に渡すために使用されます。RETARG は出力パラメーターであり、IOCTL から戻される引数を受け取るために使用されます。

REQARG と RETARG の長さと意味については、表 9 を参照してください。

表 *9. IOCTL* マクロ引数

COMMAND/コード	サイズ	**REQARG**	サイズ	**RETARG**
FIONBIO X'8004A77E'	4	ソケット・モードを次のように設定します: X'00'=blocking; X'01'=nonblocking	0	使用されません

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

LISTEN

サーバーのみが LISTEN マクロを使用します。

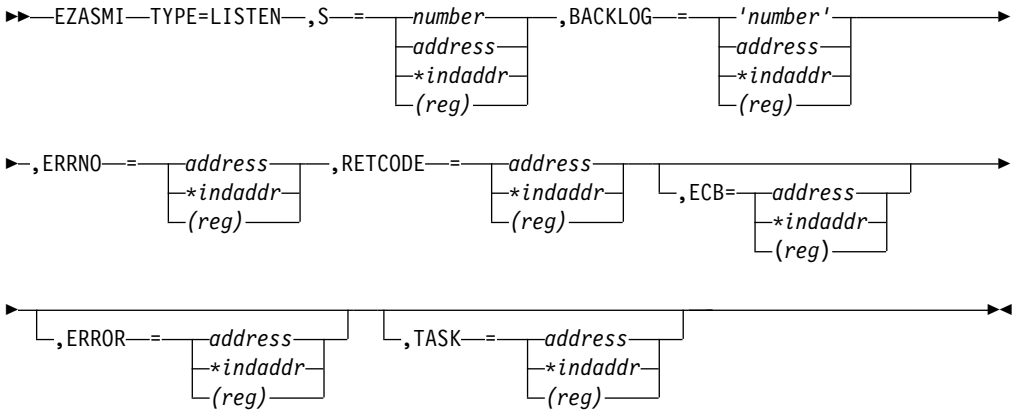
LISTEN マクロは、次のことを行います。

- クライアント接続要求を受け入れることができる作動可能状態を確立します。
- 着信する接続要求用に、指定された項目数の接続要求待ち行列を作成します。
- LISTEN マクロでは、事前に BIND 要求を発行しておく必要があります。

LISTEN マクロは、一般的に、クライアントからの接続要求を受信するために並行サーバーが使用します。接続要求が受信されると、ACCEPT マクロによって新規ソケットが作成されます。元のソケットはさらに接続要求がないかを「listen」し続けます。

注**:** 並行サーバーおよび反復サーバーが、このマクロを使用します。反復サーバーは、一時点では 1 つのクライアントを扱います。並行サーバーは複数のクライアントから接続要求を受信し、それらのクライアント要求を処理するサブタスクを作成します。サブタスクが作成されると、並行サーバーは新規ソケットを取得し、その新規ソケットをサブタスクに渡し、自身をその接続から切り離します。CICS リスナー・プログラムは、並行サーバーの 1 例です。

フォーマット



パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、ソケット記述子を指定します。

BACKLOG

入力パラメーター。値 (単一引用句で囲まれます)、またはフルワード 2 進数のアドレスであり、バックログ可能なメッセージの数を指定します。このパラメーターは無視されます。常に値 1 が想定されます。

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR

入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

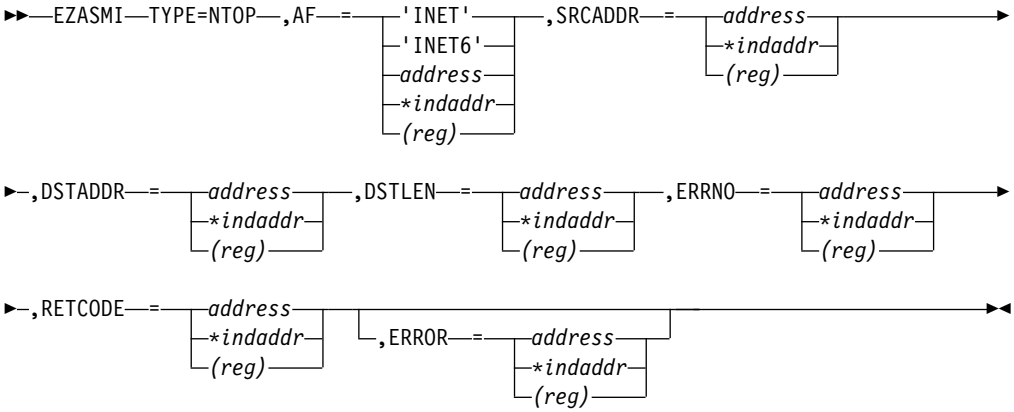
NTOP

NTOP マクロは、IP アドレスをその数値バイナリー形式から、標準テキスト表示形式に変換します。

正常に完了した場合、NTOP は変換された IP アドレスを、指定されたバッファーに返します。

重要: この関数呼び出しは TCP/IP for z/VSE では使用できません。

フォーマット



パラメーター

AF　入力パラメーター。次のいずれかを指定します。

'INET' または **10** 進数の **'2'**
変換されるアドレスが IPv4 アドレスであることを示します。

'INET6' または **10** 進数の **'19'**
変換されるアドレスが IPv6 アドレスであることを示します。

AF は、アドレス・ファミリーを指定するフルワード 2 進数を指定することもできます。

SRCADDR
入力パラメーター。変換された数値バイナリー形式の IPv4 または IPv6 アドレスが入るフィールド。IPv4 アドレスの場合、このフィールドはフルワードでなければなりません。IPv6 アドレスの場合、このフィールドは 16 バイトでなければなりません。このアドレスは、ネットワーク・バイト・オーダーでなければなりません。

DSTADDR
入力パラメーター。変換された標準テキスト表示形式の IPv4 または IPv6 アドレスを受け取るために使用するフィールド。IPv4 の場合はアドレスはドット 10 進形式で、IPv6 の場合はアドレスはコロン 16 進形式です。変換された IPv4 アドレスのサイズは最大で 15 バイトであり、変換された

IPv6 アドレスのサイズは最大で 45 バイトです。DSTADDR の値の実際の長さについては、DSTLEN で返された値を参照してください。

DSTLEN
最初は入力パラメーター。ハーフワード 2 進数フィールドのアドレスであり、入力で DSTADDR フィールドの長さを指定するために使用され、成功して戻ると、変換された IP アドレスの長さが入ります。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

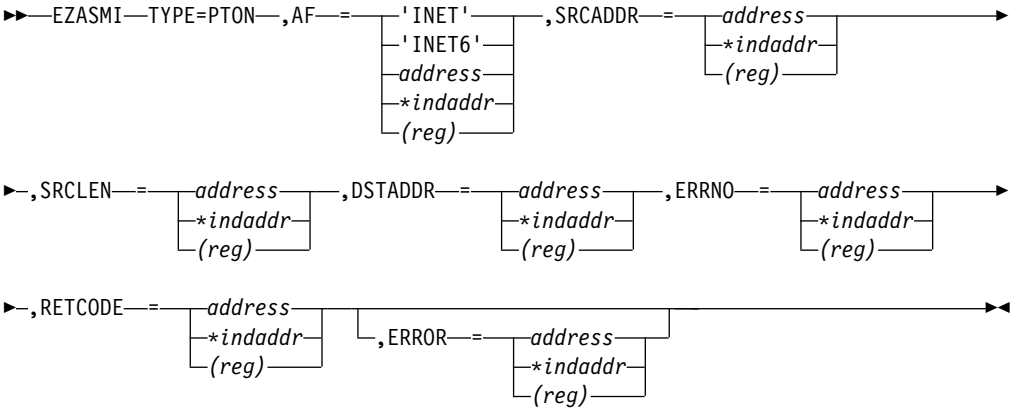
PTON

PTON マクロは、IP アドレスをその標準テキスト表示形式から数値バイナリー形式に変換します。

正常に完了した場合、PTON は変換された IP アドレスを、指定されたバッファーに返します。

重要**:** この関数呼び出しは TCP/IP for z/VSE では使用できません。

フォーマット



パラメーター

AF 入力パラメーター。次のいずれかを指定します。

'INET' または **10** 進数の **'2'**
変換されるアドレスが IPv4 アドレスであることを示します。

'INET6' または **10** 進数の **'19'**
変換されるアドレスが IPv6 アドレスであることを示します。

AF は、アドレス・ファミリーを指定するフルワード 2 進数を指定することもできます。

SRCADDR
入力パラメーター。変換された標準テキスト表示形式の IPv4 または IPv6 アドレスが入るフィールド。IPv4 の場合はアドレスはドット 10 進形式でなければならず、IPv6 の場合はアドレスはコロン 16 進形式でなければなりません。フィールドのサイズは、IPv4 アドレスの場合は 15 バイト、IPv6 アドレスの場合のサイズは 45 バイトでなければなりません。

SRCLEN
入力パラメーター。ハーフワード 2 進数フィールドであり、変換される IP アドレスの長さが入らなければなりません。

DSTADDR
ネットワーク・バイト・オーダーの、変換される数値バイナリー形式の IPv4 または IPv6 アドレスを受け取るために使用するフィールド。IPv4 アドレスの場合、このフィールドはフルワードでなければなりません。IPv6 アドレスの場合、このフィールドは 16 バイトでなければなりません。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

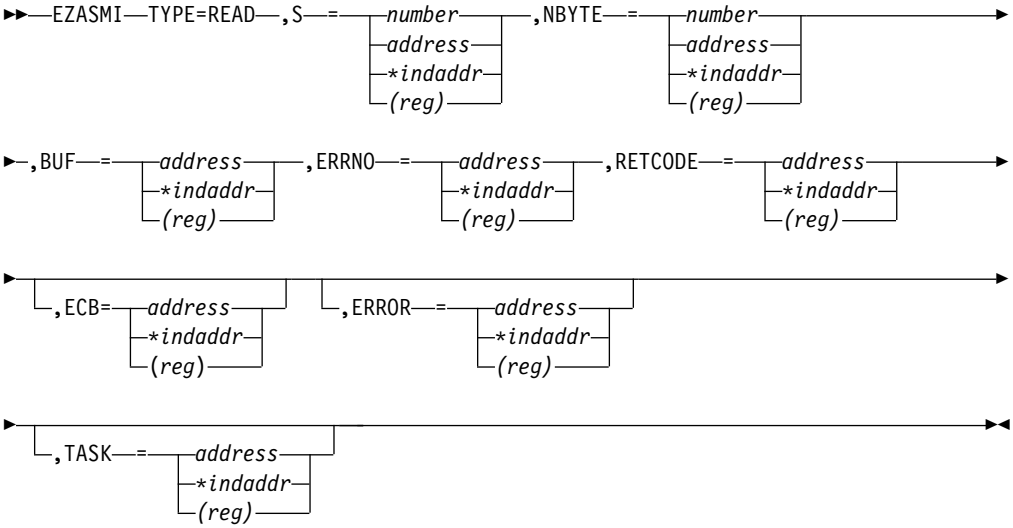
READ

READ マクロは、ソケットでデータを読み取り、そのデータをバッファーに保管します。

READ マクロは、接続されたソケットにのみ適用されます。

データグラム・ソケットの場合には、READ 呼び出しは、送信されたデータグラム全体を戻します。データグラム・パケットが長すぎて、提供されたバッファーに収まらない場合には、データグラム・ソケットは、余分なバイトを廃棄します。

フォーマット



パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、データを読み取るソケットを指定します。

NBYTE

入力パラメーター。フルワード 2 進数であり、BUF のサイズに設定されます。READ は、NBYTE に指定されたバイト数を超えるデータがある場合でも、指定のバイト数のデータだけしか戻しません。

BUF 呼び出しの完了によって埋められる入力バッファーです。BUF の長さは、最低でも NBYTE の値と同じでなければなりません。

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO フィールドは無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	戻りコード 0 は、接続がクローズしていて、使用可能なデータがないことを意味します。
>0	正の値は、バッファーにコピーされたバイト数を示します。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

READ は、NBYTE で指定されたバイト数までのデータを戻します。要求より少ないバイト数しか使用可能でない場合には、この READ マクロは、現在使用可能なバイト数を戻します。

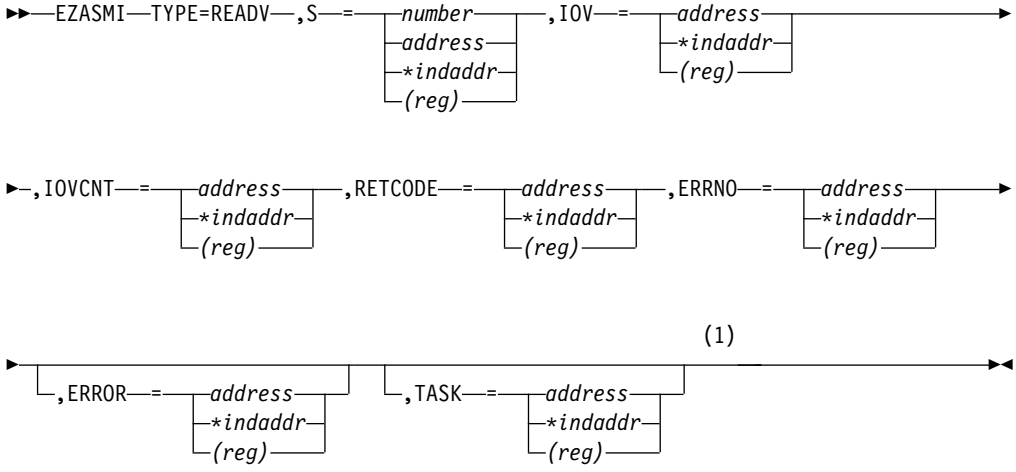
ソケットでデータが使用できず、ソケットがブロッキング・モードの場合、READ マクロはデータが到着するまで呼び出し元をブロックします。データが使用可能でなく、ソケットが非ブロッキング・モードの場合、READ は -1 を戻し、ERRNO EWOULDBLOCK を設定します。非ブロッキング・モードの設定方法の説明については、 399 ページの『IOCTL』または 349 ページの『FCNTL』を参照してください。

READV

READV マクロは、ソケットでデータを読み取り、そのデータをバッファー・セットに保管します。

データグラム・ソケットが長すぎて、提供されたバッファーに収まらない場合、余分なバイトは廃棄されます。

フォーマット



注:

1 非同期処理用の ECB パラメーターは、この呼び出しではサポートされません (z/OS とは異なります)。

パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、データを読み取るソケットを指定します。

IOV 入力パラメーター。**IOVCNT** の値と等しい数の構造体の 3 つのフルワード構造の配列。

構造体の形式は、次のとおりです。

- フルワード 1: データ・バッファーのアドレス
- フルワード 2: 予約済み
- フルワード 3: フルワード 1 で参照されるデータ・バッファーの長さ

IOVCNT
入力パラメーター。フルワードの 2 進数フィールドであり、この呼び出しに提供されるデータ・バッファー数を指定します。最大は 120 です。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	戻りコード 0 は、接続がクローズしていて、使用可能なデータがないことを意味します。
>0	バッファー・セットにコピーされたバイト数。
-1	エラーが発生しました。ERRNO のエラー・コードをチェックしてください。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO フィールドは無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

READV は、IOV 構造体のフルワードの 3 つの値の合計で指定されたバイト数までの値を戻します。使用可能なバイト数がこの合計より少ない場合、READV は現在使用可能な数を戻します。

RECV

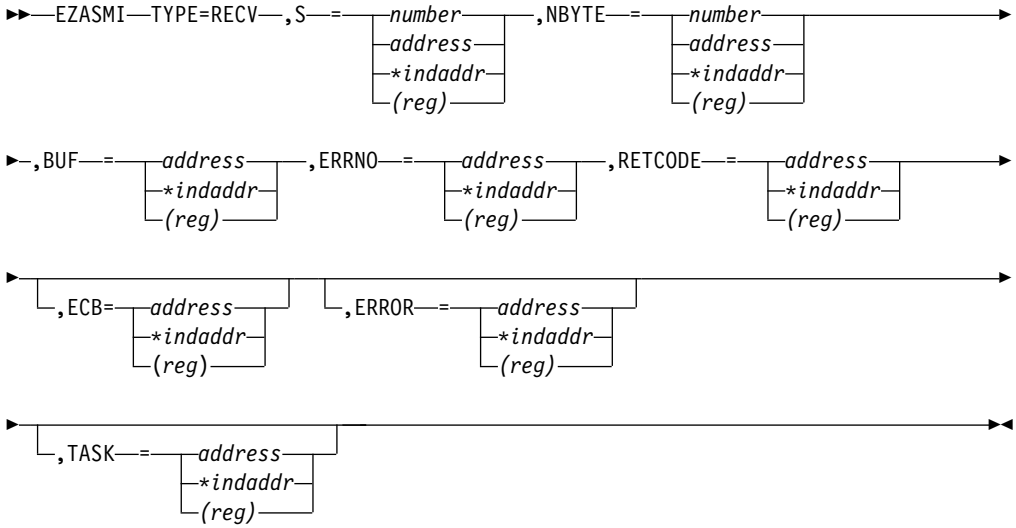
RECV マクロは、ソケットでデータを受信し、そのデータをバッファーに保管します。

RECV マクロは、接続されたソケットにのみ適用されます。

RECV は、着信メッセージまたはデータの長さを戻します。データグラム・パケットが長すぎて、提供されたバッファーに収まらない場合には、データグラム・ソケットは、余分なバイトを廃棄します。

ストリーム・ソケットの場合、データは、データを分離する境界のない情報ストリームのように処理されます。例えば、アプリケーション A および B がストリーム・ソケットと接続され、アプリケーション A が 1000 バイトを送信する場合、RECV のそれぞれの呼び出しは、1 バイト、または 10 バイト、あるいは 1000 バイト全体を戻すことができます。従って、ストリーム・ソケットを使用するアプリケーションでは、RECV をループに入れ、全データが受信されるまで呼び出しを繰り返す必要があります。

フォーマット



パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、ソケット記述子を指定します。

NBYTE
入力パラメーター。フルワード 2 進数であり、BUF のサイズに設定されます。RECV は、NBYTE に指定されたバイト数を超えるデータがある場合でも、指定のバイト数のデータだけしか受信しません。

BUF 呼び出しの完了によって埋められる入力バッファーです。BUF の長さは、最低でも NBYTE の値と同じでなければなりません。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の値の

場合、このフィールドの中にはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	戻りコード 0 は、接続がクローズしていて、使用可能なデータがないことを意味します。
>0	正の値は、バッファーにコピーされたバイト数を示します。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

ソケットでデータが使用できず、ソケットがブロッキング・モードの場合、RECV マクロはデータが到着するまで呼び出し元をブロックします。データが使用可能でなく、ソケットが非ブロッキング・モードの場合、RECV は -1 を戻し、ERRNO を EWOULDBLOCK に設定します。非ブロッキング・モードの設定方法の説明については、 349 ページの『FCNTL』または 399 ページの『IOCTL』を参照してください。

RECVFROM

RECVFROM マクロは、ソケットでデータを受信し、そのデータをバッファーに保管します。

RECVFROM は、着信メッセージまたはデータ・ストリームの長さを戻します。

記述子 S で指定されたソケットでデータが使用できず、ソケット S がブロッキング・モードの場合、RECVFROM 呼び出しはデータが到着するまで呼び出し元をブロックします。

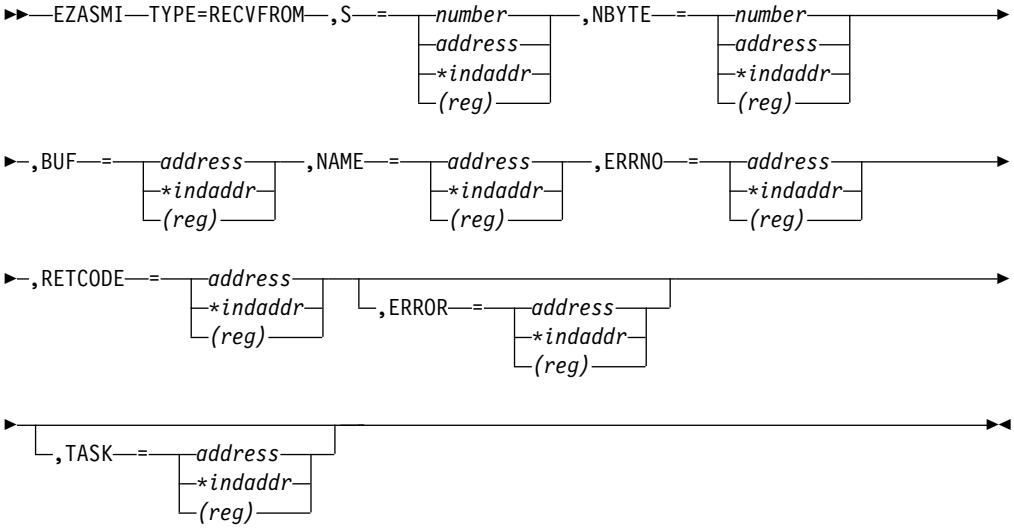
データが使用可能でなく、ソケット S が非ブロッキング・モードの場合、RECVFROM は -1 を戻し、ERRNO を EWOULDBLOCK に設定します。RECVFROM は、接続されているかどうかにかかわらずどのデータグラム・ソケットにも適用され、NAME 構造体にソケット・アドレスを戻します。非ブロッキング・モードの設定方法の説明については、 349 ページの『FCNTL』または 399 ページの

ページの『IOCTL』を参照してください。データグラム・パケットが長すぎて、提供されたバッファーに収まらない場合には、データグラム・ソケットは、余分なバイトを廃棄します。

ストリーム・ソケットの場合、データは、データを分離する境界のない情報ストリームとして処理されます。例えば、アプリケーション A および B がストリーム・ソケットと接続され、アプリケーション A が 1000 バイトを送信する場合、この関数のそれぞれの呼び出しは、1 バイト、または 10 バイト、あるいは 1000 バイト全体を戻すことができます。ストリーム・ソケットを使用するアプリケーションでは、RECVFROM をループに入れ、全データが受信されるまで繰り返す必要があります。

重要: IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

フォーマット



パラメーター

S 入力パラメーター。値、または、ハーフワード 2 進数のアドレスであり、データを受け取るソケットを指定します。

NBYTE
入力パラメーター。フルワード 2 進数のアドレスまたは値であり、入力バッファーの長さを指定します。NBYTE は、最初は NAME に関連付けられたバッファーのサイズに初期設定される必要があります。戻り時に、NBYTE には、実際に受信したデータのバイト数が入ります。

BUF 呼び出しの完了によって埋められる入力バッファーです。BUF の長さは、最低でも NBYTE の値と同じでなければなりません。

NAME
最初は、IPv4 または IPv6 アプリケーションが構造体へのポインターを設定し、呼び出しの完了時に、その構造体にピア・ソケット名が入ります。NAME パラメーター値がゼロ以外の場合は、メッセージの IPv4 または

IPv6 ソース・アドレスが挿入されます。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

FAMILY
ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、送信側ソケットのポート番号を指定します。

IPv4-ADDRESS
フルワード 2 進数フィールドであり、32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

NAMELEN
IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、送信側ソケットのポート番号を指定します。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアントのマシンの 128 ビットの IPv6 インターネット・アドレスがネットワーク・バイト・オーダーで設定されます。

SCOPE-ID
IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィー

ルド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の値の場合、このフィールドの中にはエラー番号が含まれます。ERRNO 戻りコードについては、83 ページの『ERRNO 値』 を参照してください。

RETCODE
フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0	戻りコード 0 は、接続がクローズしていて、使用可能なデータがないことを意味します。
>0	正の値は RECVFROM 呼び出しによって転送されたバイト数を示します。
-1	ERRNO のエラー・コードをチェックしてください。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

SELECT

複数の入出力操作が発生するプロセスにおいて、プログラムは 1 つまたは複数の操作の完了を待機できる必要があります。

例えば、ブロッキング・モードが設定されている複数のソケットに READ を発行するプログラムを想定します。ソケットは READ マクロでブロックするので、一度に 1 つのソケットを読みとることしかできません。ソケットを非ブロッキングに設定すると、この問題は解決できますが、データが使用可能になるまで各ソケットを繰り返しポーリングすることが必要になります。SELECT マクロを使用すると、いくつかのソケットをテストし、テストされたソケットの 1 つが作動可能である場合のみ、その後の入出力マクロを処理するようにできます。これにより、入出力マクロがブロックしないことを確実にできます。

プログラム内でタイマーとして SELECT マクロを使用するには、以下のいずれかを行います。

- 読み取り、書き込み、および例外の各配列をゼロに設定する。
- MAXSOC を指定しない。

ソケットをテストする

読み取り、書き込み、および例外の各操作をテストできます。select () 呼び出しは、選択されたソケットでのアクティビティーをモニターし、以下の項目を判別します。

- 指定されたソケット用のバッファーに入力データが含まれているかどうか。あるソケットに対する入力データがある場合、そのソケットでの読み取り操作はブロックしません。
- TCP/IP が追加出力データに対応できるかどうか。あるソケットに対して TCP/IP が追加の出力を受け入れ可能である場合、そのソケットでの書き込み操作はブロックしません。
- ソケットで例外条件が発生しているかどうか。
- SELECT マクロ自体でタイムアウトが発生しているかどうか。タイムアウト期間は、SELECT マクロの発行時に指定できます。

各ソケット記述子は、ビット・ストリング中の 1 つのビットで表されます。ビット・ストリングは、32 ビットのフルワードに含まれ、右から左に番号が付与されます。右端のビットがソケット記述子 0 を表し、左端のビットがソケット記述子 31 を表します。プロセスが 32 個以下のソケットを使用する場合、ビット・ストリングは 1 個のフルワードです。プロセスが 33 個のソケットを使用する場合、ビット・ストリングは 2 個のフルワードです。最初のフルワードがソケット記述子 0 から 31 を表し、2 番目のフルワードがソケット記述子 32 から 63 を表します。ストリング内のビットをオンにすることで、テスト対象のソケットを定義します。

読み取り操作

ACCEPT、READ、RECV、および RECVFROM マクロは、読み取り操作です。ソケットは、そのソケットでデータが受信されたとき、または、例外条件が発生したとき、読み取り可能です。

ソケットが読み取り操作可能かどうかを判別するには、RSNDMSK 中の対応するビットを '1' に設定してから、SELECT マクロを発行します。SELECT マクロが戻ると、RRETMSK 中の対応するビットによって、ソケットが読み取り可能かどうかが示されます。

書き込み操作

ソケットが書き込みに対して選択されている (書き込み可能である) ということになるのは、以下の場合です。

- TCP/IP は、追加の発信データを受け入れることができる。
- ACCEPT マクロに応じて接続要求を受け取った。
- 非ブロッキング・ソケットに対する、前に ERRNO EINPROGRESS を戻した CONNECT 呼び出しが、接続を完了した。

WRITE、SEND、または SENDTO マクロは、送信されるデータ量が TCP/IP が受け入れ可能な量を超えると、ブロックします。これを回避するには、書き込み操作の前に SELECT マクロを使用して、ソケットが書き込み可能であることを確実にします。

ソケットが書き込み操作可能かどうかを判別するには、WSNDMSK 中の対応するビットを '1' に設定します。

例外操作

テストする各ソケットについて、SELECT マクロは、例外条件があるかどうかをチェックできます。例外条件は次のとおりです。

- 呼び出し側プログラム (並行サーバー) が GIVESOCKET コマンドを発行し、ターゲットのサブタスクが正常に TAKESOCKET 呼び出しを発行した。この条件が選択されている場合、呼び出し側プログラム (並行サーバー) は、CLOSE コマンドを発行して自身をソケットから切り離すべきです。
- ソケットがアウト・オブ・バンドのデータを受信した。この条件が発生した場合、READ マクロは、アウト・オブ・バンドのデータをプログラム・データの前に戻します。

ソケットに例外条件があるかどうかを判別するには、ESNDMSK 文字ストリングを使用し、対応するビットを '1' に設定します。

結果を返す

*x*SNDMSK によってテストされる各イベントごとに、1 つのビット・ストリングに、チェックの結果が記録されます。読み取り、書き込み、例外の各イベントに関するビット・ストリングは、それぞれ RRETMSK、WRETMSK、ERETMSK です。SELECT マクロが戻るときに '1' に設定される xRETMSK 中の各ビットは、そのビットに関連するソケットでの、読み取り、書き込み、または例外のイベントを表します。

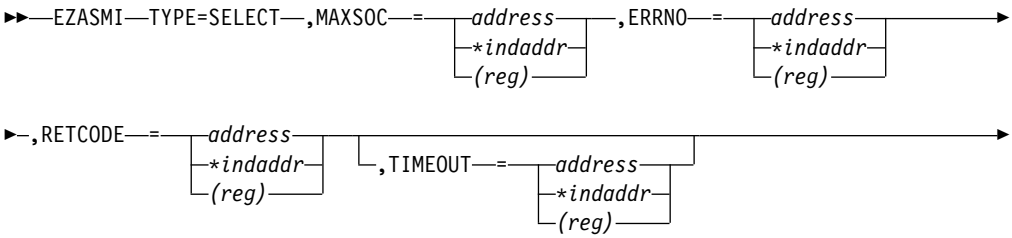
MAXSOC パラメーター

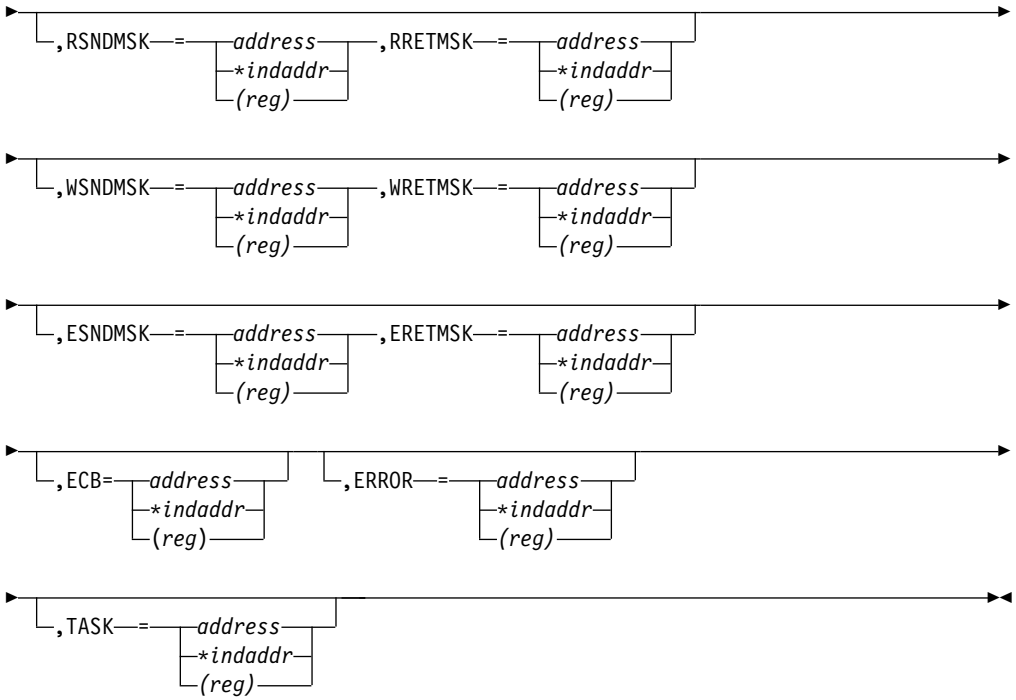
SELECT 呼び出しは、結果を戻す前に、各ストリング中の各ビットをテストしなければなりません。効率をよくするため、MAXSOC パラメーターを使用して、任意のイベント・タイプに関してソケット記述子の最大値を設定することができます。SELECT 呼び出しは、0 から MAXSOC 値までの範囲のビットだけをテストします。

TIMEOUT パラメーター

TIMEOUT パラメーターに設定された時間が経過してもイベントが何も検出されない場合、SELECT 呼び出しは、RETCODE を 0 に設定して戻ります。

フォーマット





パラメーター

MAXSOC

入力パラメーター。フルワード 2 進数フィールドであり、チェック対象のソケット記述子の最大値に 1 を加えた数値を指定します (TCP/IP for z/VSE は 0 から 8191 までの値のソケット記述子をサポートするためです)。

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
>0	3 つの戻りマスク内のすべての作動可能ソケットの数を示します。
=0	SELECT 制限時間が満了になったことを示します。
-1	ERRNO のエラー・コードをチェックしてください。

TIMEOUT

入力パラメーター。

TIMEOUT が指定されていない場合、SELECT 呼び出しは、ソケットが作動可能になるまでブロックします。

TIMEOUT が指定されている場合、TIMEOUT は、SELECT 呼び出しが最大どれだけの間隔だけ待ってから完了するのかを示します。SELECT がソケットをポーリングし、即時に戻るようにするには、ゼロに設定された TIMEVAL 構造体を指すように TIMEOUT が指定されている必要があります。

TIMEOUT は、次の 2 ワードの TIMEOUT で指定されます。

- TIMEOUT-SECONDS は、TIMEOUT の 1 番目のワードであり、タイムアウト値の秒の部分を表すコンポーネントです。
- TIMEOUT-MICROSEC は、TIMEOUT の 2 番目のワードであり、タイムアウト値のマイクロ秒の部分を表すコンポーネントです (0-999999)。

例えば、SELECT が 3.5 秒でタイムアウトになるようにするには、TIMEOUT-SECONDS を 3 に、TIMEOUT-MICROSEC を 500000 に設定します。

RSNDMSK

入力パラメーター。読み取りイベント状況を要求するために送信されるビット・ストリングです。

- 保留中の読み取りイベントをチェックする各ソケットごとに、このストリング中の対応するビットが 1 に設定される必要があります。
- 無視するソケットについては、対応するビットの値は 0 に設定される必要があります。

このパラメーターが 0 に設定されている場合、SELECT は読み取りイベントをチェックしません。このビット・マスク配列の長さは、MAXSOC の値に依存し、4 バイトの倍数でなければなりません。

RRETMSK

出力パラメーター。読み取りイベントの状況を戻すビット・ストリングです。

- 読み取りに可能な各ソケットについては、このストリング中の対応するビットが 1 に設定されます。
- 無視されるソケットについては、このストリング中の対応するビットは 0 に設定されます。

WSNDMSK

入力パラメーター。書き込みイベント状況を要求するために送信されるビット・ストリングです。

- 保留中の書き込みイベントをチェックする各ソケットごとに、このストリング中の対応するビットが 1 に設定される必要があります。
- 無視するソケットについては、対応するビットの値は 0 に設定される必要があります。

WRETMSK

出力パラメーター。書き込みイベントの状況を戻すビット・ストリングです。

- 書き込み可能な各ソケットについては、このストリング中の対応するビットが 1 に設定されます。

- 書き込み可能でない各ソケットについては、このストリング中の対応するビットが 0 に設定されます。

ESNDMSK
入力パラメーター。例外イベント状況を要求するために送信されるビット・ストリングです。このストリングの長さは、チェック対象のソケットの最大数と同じである必要があります。

- 保留中の例外イベントをチェックする各ソケットごとに、このストリング中の対応するビットが 1 に設定される必要があります。
- 無視するソケットについては、対応するビットは 0 に設定される必要があります。

ERETMSK
出力パラメーター。例外イベントの状況を戻すビット・ストリングです。このストリングの長さは、チェック対象のソケットの最大数と同じである必要があります。

- 例外状況が設定された各ソケットについては、このストリング中の対応するビットが 1 に設定されます。
- 例外状況のない各ソケットについては、対応するビットが 0 に設定されます。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注: このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

SELECTEX

SELECTEX マクロは、ソケットの集合、タイム値、1 つの ECB または ECB リストをモニターします。

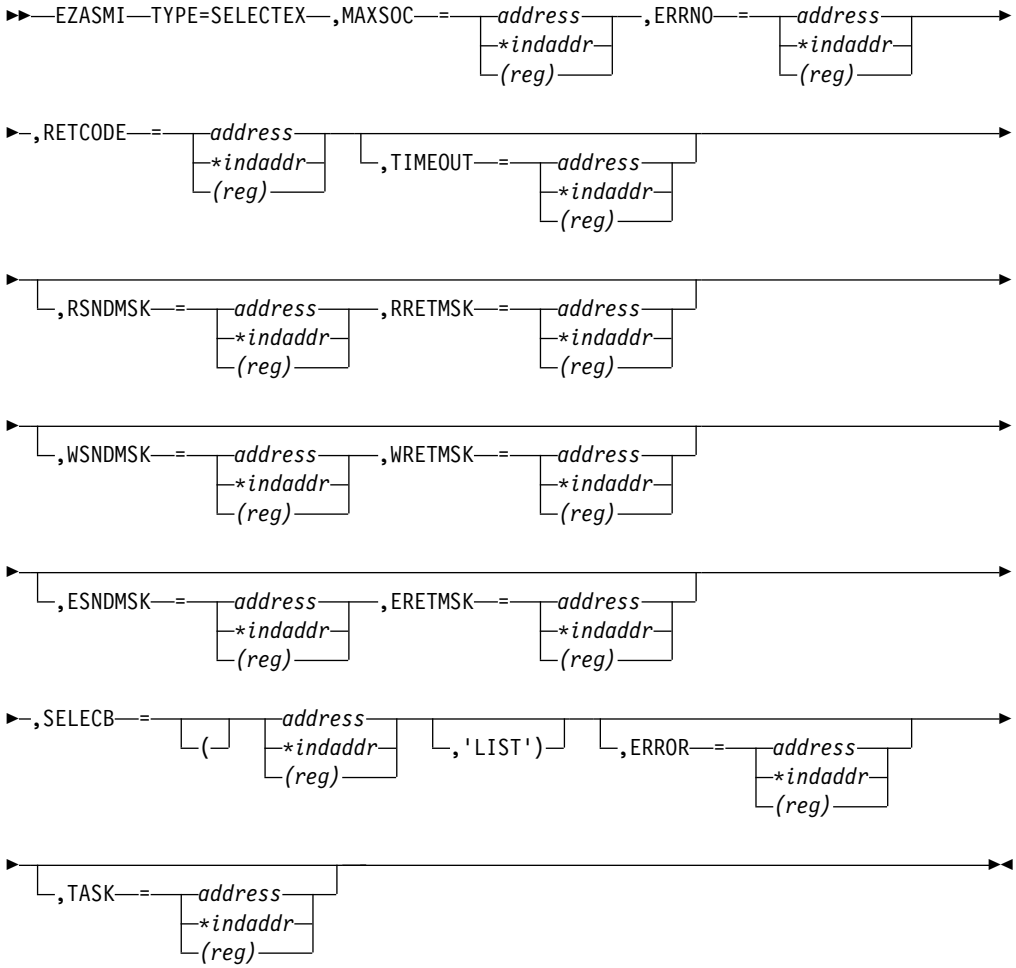
これが完了するのは、ソケットの 1 つにアクティビティーがあるとき、タイム値が満了したとき、または、ECB がポストされたときのいずれかです。

プログラム内でタイマーとして SELECTEX 呼び出しを使用するには、以下のいずれかを行います。

- 読み取り、書き込み、および例外の各配列をゼロに設定する。
- MAXSOC を指定しない。

ソケットのテストについての詳しい説明は、 413 ページの『SELECT』を参照してください。

フォーマット



パラメーター

MAXSOC

入力パラメーター。フルワード 2 進数フィールドであり、チェック対象のソケット記述子の最大値に 1 を加えた数値を指定します (TCP/IP for z/VSE は 0 から 8191 までの値のソケット記述子をサポートするためです)。

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の値の場合、このフィールドの中にはエラー番号が含まれます。

RETCODE

出力パラメーター。フルワード 2 進数フィールド。

値	説明
>0	作動可能ソケットの数を示します。
0	SELECTEX の時間制限が満了した (ECB 値は 0 になります) か、

呼び出し元の ECB の 1 つがポストされた (ECB 値は非ゼロになり、呼び出し元の記述子セットは 0 に設定されます) かのいずれかです。呼び出し元は、SELECTEX マクロを発行する前に、ECB 値をゼロに初期化しなければなりません。

-1 ERRNO をチェックしてください。

TIMEOUT

入力パラメーター。

TIMEOUT が指定されていない場合、SELECTEX 呼び出しは、ソケットが作動可能になるか、またはユーザー ECB がポストされるまでブロックします。

TIMEOUT 値が指定されている場合、TIMEOUT は、SELECTEX 呼び出しが最大どれだけの間隔だけ待ってから完了するのかを示します。SELECTEX がソケットをポーリングし、即時に戻るようにするには、ゼロに設定された TIMEVAL 構造体を指すように TIMEOUT が指定されている必要があります。

TIMEOUT は、次の 2 ワードの TIMEOUT で指定されます。

- TIMEOUT-SECONDS は、TIMEOUT の 1 番目のワードであり、タイムアウト値の秒の部分を表すコンポーネントです。
- TIMEOUT-MICROSEC は、TIMEOUT の 2 番目のワードであり、タイムアウト値のマイクロ秒の部分を表すコンポーネントです (0—999999)。

例えば、SELECT が 3.5 秒でタイムアウトになるようにするには、TIMEOUT-SECONDS を 3 に、TIMEOUT-MICROSEC を 500000 に設定します。タイムアウトになると、SELECTEX は呼び出し側プログラムに戻ります。

RSNDMSK

入力パラメーター。読み取り割り込みのチェックを制御するビット・マスク配列です。このパラメーターが指定されていないか、指定されたビット・マスクがゼロである場合、SELECT は読み取り割り込みをチェックしません。このビット・マスク配列の長さは、MAXSOC の値に依存し、4 バイトの倍数でなければなりません。

RRETMSK

出力パラメーター。RSNDMSK が指定されている場合に SELECT が戻すビット・マスク配列です。このビット・マスク配列の長さは、MAXSOC の値に依存し、4 バイトの倍数でなければなりません。

WSNDMSK

入力パラメーター。書き込み割り込みのチェックを制御するビット・マスク配列です。このパラメーターが指定されていないか、指定されたビット・マスクがゼロである場合、SELECT は書き込み割り込みをチェックしません。このビット・マスク配列の長さは、MAXSOC の値に依存し、4 バイトの倍数でなければなりません。

WRETMSK

出力パラメーター。**WSNDMSK** が指定されている場合に SELECT が戻すビット・マスク配列です。このビット・マスク配列の長さは、MAXSOC の値に依存し、4 バイトの倍数でなければなりません。

ESNDMSK
入力パラメーター。例外割り込みのチェックを制御するビット・マスク配列です。このパラメーターが指定されていないか、指定されたビット・マスクがゼロである場合、SELECT は例外割り込みをチェックしません。このビット・マスク配列の長さは、MAXSOC の値に依存し、4 バイトの倍数でなければなりません。

ERETMSK
出力パラメーター。ESNDMSK が指定されている場合に SELECT が戻すビット・マスク配列です。このビット・マスク配列の長さは、MAXSOC の値に依存し、4 バイトの倍数でなければなりません。

SELECB
入力パラメーター。ポストされると SELECTEX を完了させることになる、1 つの ECB または、複数の ECB アドレスからなるリストです。

ECB アドレス・リストのアドレスが指定される場合は、その ECB リストの最終項目の上位ビットを 1 に設定する必要があり、さらに、LIST キーワードを追加する必要があります。ECB は、呼び出し元のホーム・アドレス・スペース内になければなりません。

注: リスト中に指定できる ECB の最大数は 254 です。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

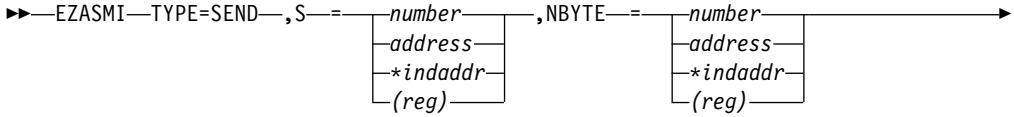
SEND

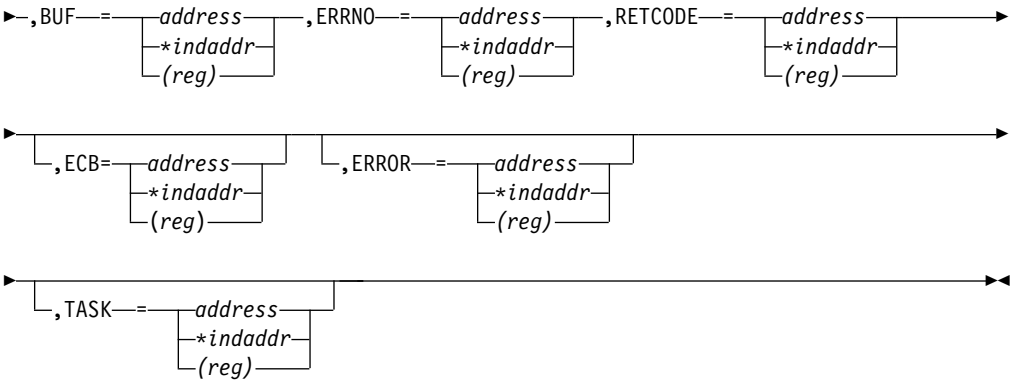
SEND マクロは、指定された接続済みソケット上でデータグラムを送信します。

データグラム・ソケットの場合、SEND はデータグラム全体を、それが受信バッファーに収まる場合は、送信します。余分なデータは廃棄されます。

ストリーム・ソケットの場合、データは、データを分離する境界のない情報ストリームとして処理されます。例えば、プログラムが 1000 バイトを送信する必要がある場合、この関数のそれぞれの呼び出しは、1000 バイト全部までの任意のバイト数を送信することができ、送信されたバイト数が **RETCODE** に入れて戻されます。従って、ストリーム・ソケットを使用するプログラムでは、この呼び出しをループに入れ、全データが送信されるまでこの呼び出しを繰り返し発行する必要があります。

フォーマット





パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、データを送信するソケットのソケット記述子を指定します。

NBYTE
入力パラメーター。フルワード 2 進数のアドレスまたは値であり、送信するバイト数を指定します。

BUF 送信されるデータのアドレスです。BUF の長さは、最低でも NBYTE の値と同じでなければなりません。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の値の場合、このフィールドの中にはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールド。

値 説明

0 または **>0**
呼び出しは成功しました。値は、送信されたバイト数に設定されます。

-1 **ERRNO** のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

SENDTO

SENDTO は、宛先アドレス・パラメーターを指定することを除いて SEND と同様です。

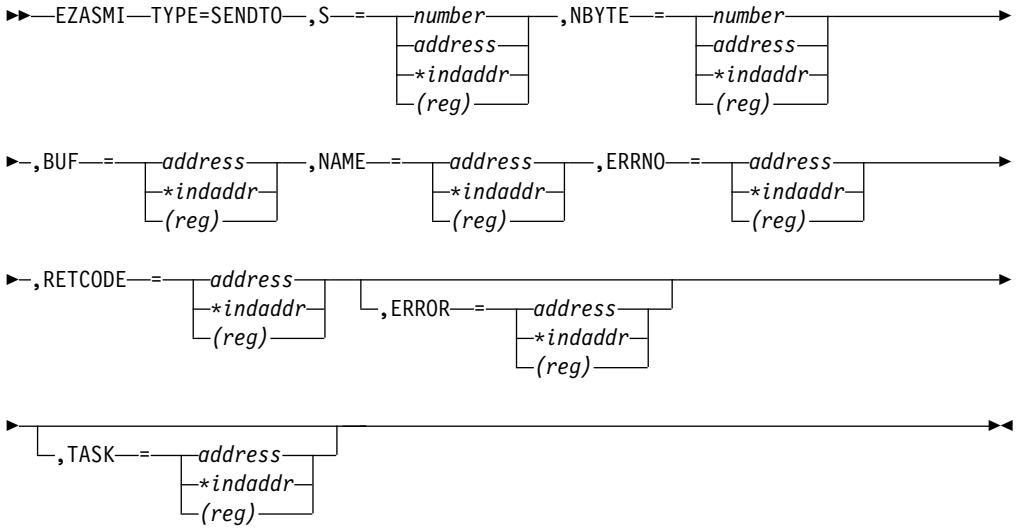
SENDTO マクロで宛先アドレスを使用すると、接続されている、または接続されていない UDP ソケットでデータグラムを送信することができます。

データグラム・ソケットの場合、データグラムがバッファーに入りきれば、SENDTO マクロはデータグラム全体を送信します。

ストリーム・ソケットの場合、データは、データを分離する境界のない情報ストリームとして処理されます。例えば、プログラムが 1000 バイトを送信する必要がある場合、それぞれの SENDTO マクロ呼び出しは、1000 バイト全部までの任意のバイト数を送信することができ、送信されたバイト数が RETCODE に入れて戻されます。従って、ストリーム・ソケットを使用するプログラムでは、SENDTO をループに入れ、全データが送信されるまでマクロを繰り返す必要があります。

重要: IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

フォーマット



パラメーター

S 出力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、データを送信するソケットのソケット記述子を指定します。

NBYTE
入力パラメーター。フルワード 2 進数のアドレスまたは値であり、送信するバイト数を指定します。

BUF 入力パラメーター。送信されるデータのアドレスです。BUF の長さは、最低でも NBYTE の値と同じでなければなりません。

NAME

入力パラメーター。IPv4 または IPv6 ターゲットのアドレス。PRD1.MACLIB(EZBREHST) マクロを含めて、ソケット・アドレス構造体のアセンブラー・マッピングを取得します。ソケット・アドレス構造体のマッピングは、SOCKADDR ラベルで開始します。AF_INET ソケット・アドレス構造体のフィールドは、SOCK_SIN ラベルで開始します。AF_INET6 ソケット・アドレス構造体のフィールドは、SOCK_SIN6 ラベルで開始します。

IPv4 ソケット・アドレス構造体は、以下のフィールドを含みます。

FAMILY
ハーフワード 2 進数フィールドであり、IPv4 アドレス・ファミリーを指定します。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、ソケットにバインドされたポート番号を指定します。

IPv4-ADDRESS
フルワード 2 進数フィールドであり、ソケットの 32 ビット IPv4 インターネット・アドレスを指定します。

RESERVED
2 進ゼロの 8 バイトを指定します。このフィールドは、使用されません。

IPv6 ソケット・アドレス構造体は、以下のフィールドを含みます。

NAMELEN
IPv6 ソケット・アドレス構造体の長さを指定する、1 バイトの 2 進数フィールド。このフィールドの使用により指定されたいかなる値も、入力として処理される場合には無視されます。出力として処理される場合には、フィールドは 0 に設定されます。

FAMILY
IPv6 アドレス・ファミリーを指定する、1 バイトの 2 進数フィールド。TCP/IP の場合、この値は、AF_INET6 を示す 10 進数の 19 に設定されます。

PORT
ハーフワード 2 進数フィールドであり、ソケットにバインドされたポート番号を指定します。

FLOW-INFO
フルワード 2 進数フィールドであり、トラフィック・クラスおよびフロー・ラベルを指定します。このフィールドのこの値は定義されていません。

IPv6-ADDRESS
16 バイト 2 進数フィールドであり、クライアント・マシンのソケットの 128 ビット IPv6 インターネット・アドレスが、ネットワーク・バイト・オーダーで設定されます。

SCOPE-ID

IPv6-ADDRESS フィールドに入れるアドレスのスコープとして適切なインターフェースのセットを識別する、フルワード 2 進数フィールド。リンク・スコープ IPv6-ADDRESS では、SCOPE-ID は IPv6-ADDRESS のリンク・インデックスを含みます。その他すべてのアドレス・スコープでは、SCOPE-ID は定義されていません。

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
0 または **>0**	呼び出しは成功しました。値は、送信されたバイト数に設定されます。
-1	ERRNO のエラー・コードをチェックしてください。

ERROR

入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK

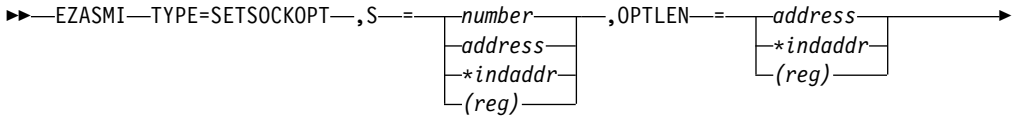
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

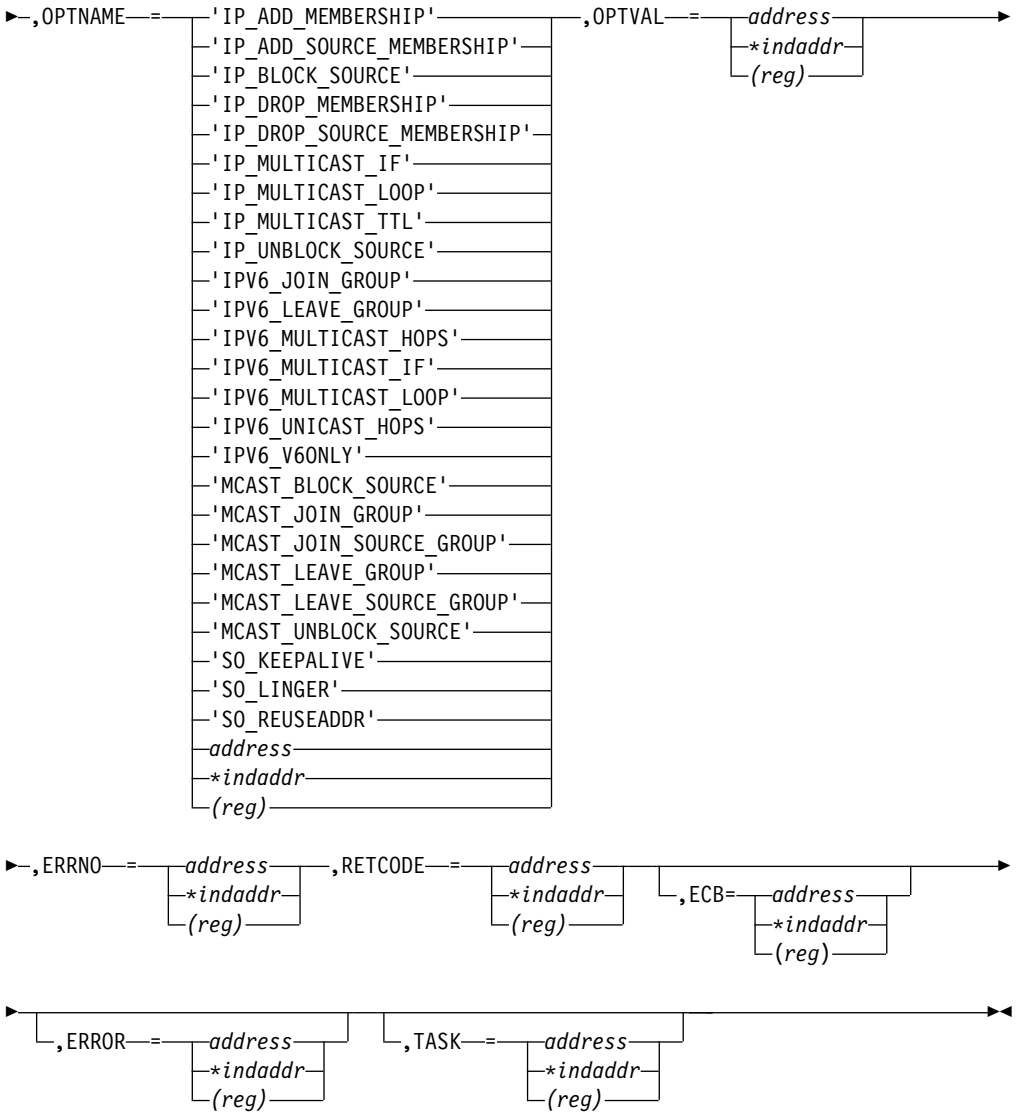
SETSOCKOPT

SETSOCKOPT マクロは、ソケットに関連付けられたオプションを設定します。

OPTVAL パラメーターと **OPTLEN** パラメーターは、特定の設定コマンドで使用されるデータを渡すために使用されます。**OPTVAL** パラメーターは、設定コマンドが必要とするデータを含むバッファーを指します。**OPTLEN** パラメーターは、**OPTVAL** によって示されるデータのサイズに設定する必要があります。

フォーマット





パラメーター

S ハーフワード 2 進数のアドレスまたは値であり、データを送信するソケットを指定します。

OPTLEN

入力パラメーター。フルワード 2 進数であり、**OPTVAL** で指定されるフィールドの長さを指定します。

OPTNAME

入力パラメーター。以下の値を指定します。

IP_ADD_MEMBERSHIP

アプリケーションが特定のインターフェース上のマルチキャスト・グループに参加できるようにするには、このオプションを使用します。このオプションでは、インターフェースを指定する必要があります。マルチキャスト・データグラムを受信するアプリケーションのみが、マルチキャスト・グループを結合する必要があります。これは IPv4 専用のソケット・オプションです。

IP_ADD_SOURCE_MEMBERSHIP
アプリケーションが特定のインターフェース上の特定のソース・アドレスを持つソース・マルチキャスト・グループに参加できるようにするには、このオプションを使用します。このオプションでは、インターフェースとソース・アドレスを指定する必要があります。マルチキャスト・データグラムを受け取るためには、アプリケーションがソース・マルチキャスト・グループに参加している必要があります。これは IPv4 専用のソケット・オプションです。

IP_BLOCK_SOURCE
指定された IPv4 ソース・アドレスと一致するソース・アドレスを持つマルチキャスト・パケットをアプリケーションでブロックできるようにするには、このオプションを使用します。このオプションでは、インターフェースとソース・アドレスを指定する必要があります。指定されたマルチキャスト・グループにあらかじめ参加しておく必要があります。これは IPv4 専用のソケット・オプションです。

IP_DROP_MEMBERSHIP
アプリケーションがマルチキャスト・グループまたはマルチキャスト・グループのすべてのソースから離脱できるようにするには、このオプションを使用します。これは IPv4 専用のソケット・オプションです。

IP_DROP_SOURCE_MEMBERSHIP
アプリケーションがソース・マルチキャスト・グループから離脱できるようにするには、このオプションを使用します。これは IPv4 専用のソケット・オプションです。

IP_MULTICAST_IF
このオプションは、ソケット・アプリケーションからアウトバウンド・マルチキャスト・データグラムを送信する際に使用する IPv4 インターフェース・アドレスを設定するために使用します。これは IPv4 専用のソケット・オプションです。

注**:** マルチキャスト・データグラムを送信できるインターフェースは、一度に 1 つのみです。

IP_MULTICAST_LOOP
送信ホスト自体が属するグループに送信されたマルチキャスト・データグラムについて、マルチキャスト・データグラムのコピーをループバックするかどうかを制御するには、このオプションを使用します。デフォルトでは、データグラムをループバックします。これは IPv4 専用のソケット・オプションです。

IP_MULTICAST_TTL
このオプションを使用して、発信マルチキャスト・データグラムの IP データグラム存続時間 (ホップ) を設定します。デフォルト値は '01'x です。これは、マルチキャストがローカル・サブネットでのみ使用可能であることを意味します。これは IPv4 専用のソケット・オプションです。

IP_UNBLOCK_SOURCE
このオプションは、所定の IPv4 マルチキャスト・グループに対して以前にブロックしたソースをアプリケーションでブロック解除できるようにするために使用します。このオプションでは、インターフェースとソース・アドレスを指定する必要があります。これは IPv4 専用のソケット・オプションです。

IPV6_JOIN_GROUP
このオプションを使用して、マルチキャスト・パケットの受信を制御し、ソケットがマルチキャスト・グループに参加することを指定します。これは IPv6 専用のソケット・オプションです。

IPV6_LEAVE_GROUP
このオプションを使用して、マルチキャスト・パケットの受信を制御し、ソケットがマルチキャスト・グループから離脱することを指定します。これは IPv6 専用のソケット・オプションです。

IPV6_MULTICAST_HOPS
発信マルチキャスト・パケットに使用するホップ限界を設定するために使用します。これは IPv6 専用のソケット・オプションです。

IPV6_MULTICAST_IF
このオプションを使用して、ソケット・アプリケーションからアウトバウンド・マルチキャスト・データグラムを送信する際に使用する IPv6 インターフェースの索引を設定します。これは IPv6 専用のソケット・オプションです。

IPV6_MULTICAST_LOOP
送信ホスト自体が属するグループに対してデータグラムを送信する場合、ローカル配信のため、IP レイヤーによって発信インターフェース上でマルチキャスト・データグラムをループバックするかどうかを制御するには、このオプションを使用します。デフォルトでは、マルチキャスト・データグラムをループバックします。これは IPv6 専用のソケット・オプションです。

IPV6_UNICAST_HOPS
このオプションを使用して、発信ユニキャスト IPv6 パケットに使用するホップ限界を設定します。これは IPv6 専用のソケット・オプションです。

IPV6_V6ONLY
このオプションを使用して、ソケットを IPv6 パケットのみの送受信に制限するかどうかを設定します。デフォルトでは、IPv6 パケットのみの送受信に制限しません。これは IPv6 専用のソケット・オプションです。

MCAST_BLOCK_SOURCE
所定のソース・アドレスと一致するソース・アドレスを持つマルチキャスト・パケットをアプリケーションでブロックできるようにするには、このオプションを使用します。このオプションでは、インターフェース索引とソース・アドレスを指定する必要があります。指定されたマルチキャスト・グループにあらかじめ参加しておく必要があります。

MCAST_JOIN_GROUP
アプリケーションが特定のインターフェース上のマルチキャスト・グループに参加できるようにするには、このオプションを使用します。インターフェース索引を指定する必要があります。マルチキャスト・データグラムを受け取るには、アプリケーションがマルチキャスト・グループに参加する必要があります。

MCAST_JOIN_SOURCE_GROUP
アプリケーションが特定のインターフェースおよびソース・アドレスにあるソース・マルチキャスト・グループに参加できるようにするには、このオプションを使用します。インターフェース索引とソース・アドレスを指定する必要があります。特定のソース・アドレスからのみマルチキャスト・データグラムを受け取るには、アプリケーションがソース・マルチキャスト・グループに参加する必要があります。

MCAST_LEAVE_GROUP
アプリケーションがマルチキャスト・グループまたは所定のマルチキャスト・グループのすべてのソースから離脱できるようにするには、このオプションを使用します。

MCAST_LEAVE_SOURCE_GROUP
アプリケーションがソース・マルチキャスト・グループから離脱できるようにするには、このオプションを使用します。

MCAST_UNBLOCK_SOURCE
このオプションは、所定のマルチキャスト・グループに対して以前にブロックしたソースをアプリケーションでブロック解除できるようにするために使用します。このオプションでは、インターフェース索引とソース・アドレスを指定する必要があります。

SO_KEEPALIVE
このオプションは、ソースの互換性のためにのみ用意されています。何もアクションは行いません。この代わりに、ユーザーは一般的 TCP/IP 設定である SET PULSE_TIME=nnn を使用するべきです。

SO_LINGER
ソケットに対して CLOSE マクロが発行されたとき、送信できなかったデータを TCP/IP がどのように扱うのかを制御します。このオプションは、ストリーム・ソケットにのみ意味を持ちます。

- **SO_LINGER** が設定されていて、CLOSE が呼び出される場合、呼び出し側プログラムは、データが正常に送信されるか、接続がタイムアウトになるまでブロックされます。
- SO_LINGER が設定されていない場合、CLOSE マクロは呼び出し元をブロックせずに戻り、TCP/IP は、指定された時間だけ引き続きデータを送信しようとします。通常、これはデータ送信を完了するのに十分な時間です。SO_LINGER オプションを使用すると、TCP/IP が待つのは、SO_LINGER に対して **OPTVAL** に指定された時間だけなので、正常に完了することは保証されません。

デフォルトでは使用不可に設定されます。

SO_REUSEADDR
このオプションは、ソースの互換性のためにのみ用意されています。何もアクションは行いません。TCP/IP は、暗黙に、アドレスの即時再使用を許可しています。

OPTVAL
入力パラメーター。**OPTNAME** に指定されたオプションに関するデータを含みます。

- **OPTVAL** は、**OPTNAME** 値が SO_LINGER でない場合は、32 ビットの 2 進数です。**OPTVAL** をゼロ以外の正の値に設定すると、オプションが使用可能に設定されます。**OPTVAL** をゼロに設定すると、オプションは使用不可になります。
- SO_LINGER の場合、**OPTVAL** は次のとおりです。

```
ONOFF    DS   F          ON OR OFF
LINGER   DS   F          TIME IN SECONDS
```

ONOFF をゼロ以外の値に設定するとオプションが使用可能に設定され、ゼロに設定するとオプションは使用不可になります。LINGER 値は、CLOSE マクロが発行された後に TCP/IP がデータ送信を続行する時間を秒単位で設定します。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
>0	3 つの戻りマスク内のすべての作動可能ソケットの数を示します。
=0	SELECT 制限時間が満了になったことを示します。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

OPTVAL パラメーターと OPTLEN パラメーターは、特定の設定コマンドで使用されるデータを渡すために使用されます。OPTVAL パラメーターは、設定コマンドが必要とするデータを含むバッファーを指します。このパラメーターはオプションであり、コマンドがデータを必要としない場合は NULL ポインターに設定できます。OPTLEN パラメーターは、**OPTVAL** によって示されるデータのサイズに設定する必要があります。

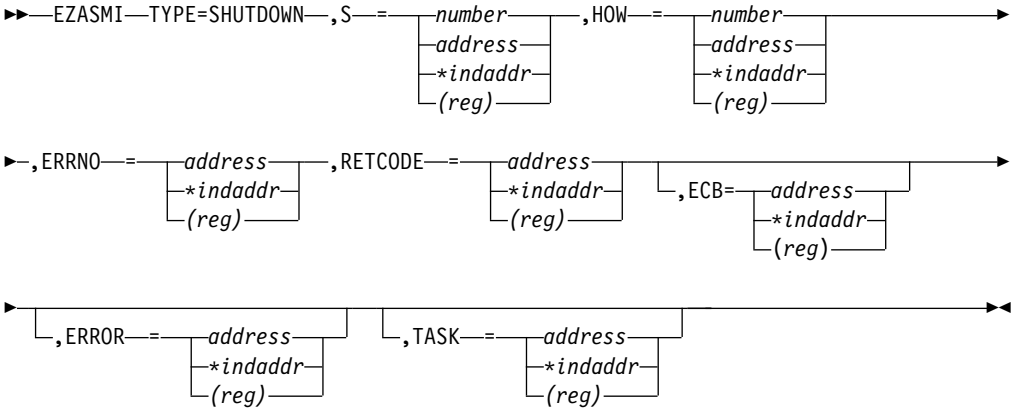
SHUTDOWN

SHUTDOWN マクロは、片方向のトラフィックをクローズさせる一方で、他方向のデータ転送を完了させるために使用できます。

HOW パラメーターが、シャットダウンするトラフィックの方向を指定します。クライアント・プログラムは、SHUTDOWN マクロを使うことにより、特定のソケットを異なる接続で再使用できるようにします。

ネットワーク接続を終了させるもう 1 つの方法は 345 ページの『CLOSE』マクロを発行することです。このマクロは、接続を切断する前に、未処理のすべてのデータ伝送要求を完了させようとします。

フォーマット



パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、シャットダウンされるソケットを指定します。

HOW 入力パラメーター。フルワード 2 進数フィールドであり、シャットダウン方式を指定します。

値	説明
2	それ以上の送信および受信操作を終了します。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下の値を戻します。

値	説明
0	呼び出しは成功しました。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、**ECB** がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

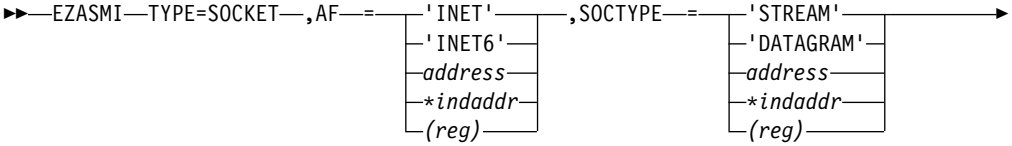
SOCKET

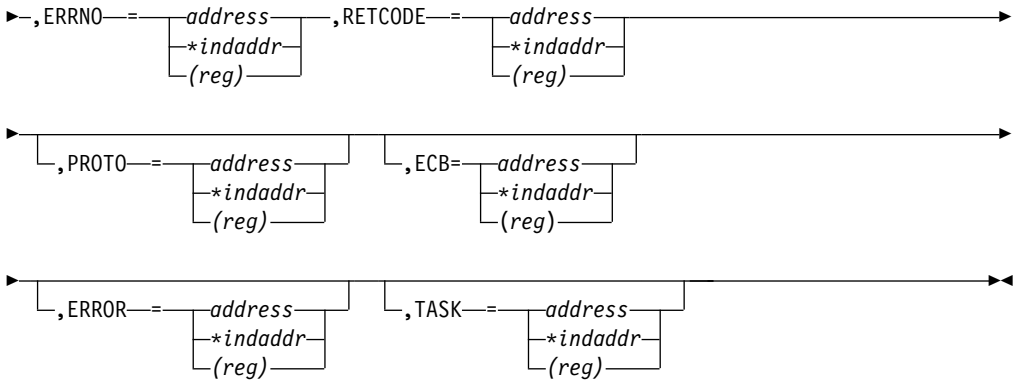
SOCKET マクロは通信の端点を作成し、その端点を表すソケット記述子を戻します。

異なるタイプのソケットで、異なる通信サービスが提供されます。

重要**:** IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

フォーマット





パラメーター

AF 入力パラメーター。インターネットまたは TCP/IP を示すリテラル INET または INET6 を指定します。次のいずれかを指定します。

'INET' または **10** 進数の **'2'**
変換されるアドレスが IPv4 アドレスであることを示します。

'INET6' または **10** 進数の **'19'**
変換されるアドレスが IPv6 アドレスであることを示します。

AF は、アドレス・ファミリーを指定するフルワード 2 進数を指定することもできます。

SOCTYPE
入力パラメーター。フルワード 2 進数フィールドであり、必要なソケットのタイプに設定されます。タイプは、以下のとおりです。

1 または **'STREAM'**
ストリーム・ソケットは、信頼性があり、コネクション指向の、順次両方向バイト・ストリームを提供します。アウト・オブ・バンドのデータのメカニズムがサポートされます。これは、TCP/IP の標準タイプです。

2 または **'DATAGRAM'**
データグラム・ソケットは、信頼性が保証されていない、固定最大長のコネクションレス・メッセージであるデータグラムを提供します。データグラムでは、破壊、順序が狂った受信、紛失、または複数回の送達が起こる場合があります。このタイプは、AF_INET ドメインでのみサポートされます。

注**:** ロー・ソケットはサポートされません

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の値の場合、このフィールドの中にはエラー番号が含まれます。ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値　　説明

> または **= 0**
新しいソケット記述子を含みます。

-1 ERRNO のエラー・コードをチェックしてください。

PROTO
入力パラメーター。フルワードの 2 進数であり、サポートされるプロトコルを指定します。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

PROTO は、ソケットで使用される特定のプロトコルを指定します。PROTO が 0 に設定されている場合、要求されたドメインおよびソケット・タイプに対するデフォルトのプロトコル番号がシステムによって選択されます。PROTO のデフォルトは、ストリーム・ソケットの場合は TCP、データグラム・ソケットの場合は UDP です。PROTO が 17 に設定されている場合、UDP プロトコルが使用されます。6 に設定されている場合、TCP プロトコルが使用されます。

SOCK_STREAM ソケットは、二重バイト・ストリームをモデル化したものです。ピア・アプリケーション間の、信頼性のあるフロー制御接続を提供します。ストリーム・ソケットは、能動または受動のいずれかです。能動ソケットは、CONNECT を使用して接続要求を開始するクライアントによって使用されます。デフォルトでは、SOCKET は能動ソケットを作成します。受動ソケットは、CONNECT マクロによる接続要求を受け入れるために、サーバーによって使用されます。能動ソケットは、BIND マクロによってソケットに名前を結合し、LISTEN マクロによって接続を受け入れることを示すことにより、受動ソケットに変換することができます。ソケットが受動になった後には、接続要求を開始するためにそのソケットを使用することはできません。

AF_INET ドメインでは、ストリーム・ソケットに適用される BIND マクロによって、アプリケーションは、接続要求を受け入れるネットワークを指定することができます。アプリケーションは、アドレス構造体の IP アドレス・フィールドをネットワーク・インターフェースの IP アドレスに設定することによって、ネットワーク・アドレスを完全に指定できます。または、アプリケーションは、名前構造体のアドレスをゼロに設定することによって、どのネットワークからでも接続要求を受け取ることを指定できます。

ストリーム・ソケット間でいったん接続が確立されると、データ転送マクロ READ、WRITE、SEND、RECV、SENDTO、および RECVFROM を使用できます。通常は、READ-WRITE または SEND-RECV ペアが、ストリーム・ソケットでのデータの送信に使用されます。

SOCK_DGRAM ソケットは、データグラムをモデル化するのに使用されます。信頼性の保証がない、コネクションレス・メッセージ交換を提供します。送信メッセージのサイズは最大です。

ストリーム・ソケットでの能動または受動という概念は、データグラム・ソケットには当てはまりません。サーバーは、BIND を呼び出してソケットに名前を付け、どのネットワーク・インターフェースからデータグラムを受信するのかを指定しなければなりません。ストリーム・ソケットで説明したワイルドカードによるアドレッシングは、データグラム・ソケットにも適用されます。データグラム・ソケットはコネクションレスなので、LISTEN マクロは、これらにとっては無意味で、使用する必要もありません。

アプリケーションは、データグラム・ソケットを受け取った後、SENDTO マクロと RECVFROM マクロを使用して、データグラムを交換できます。アプリケーションが、CONNECT を呼び出し、すべてのメッセージの交換に使用されるピアの名前を完全に指定することによって、さらに 1 ステップ先に進むと、他のデータ転送マクロ READ、WRITE、SEND、RECV も使用できます。ソケットを接続状態にすることに関する説明については、 239 ページの『CONNECT』を参照してください。

データグラム・ソケットを使用して、複数の宛先にメッセージをブロードキャストすることができます。宛先アドレスをブロードキャスト・アドレスに設定する方法は、ネットワーク・インターフェース (アドレス・クラス、およびサブネット使用の有無) によって異なります。

発信データグラムには、IP ヘッダーが接頭部として付けられています。プログラムは、IP ヘッダーが付いたままの着信データグラムを受信します。SETSOCKOPT マクロと GETSOCKOPT マクロを使用して、IP オプションの設定および検査を行うことができます。

ソケットを割り振り解除するには、CLOSE マクロを使用します。

TAKESOCKET

TAKESOCKET マクロは他のプログラムからソケットを取得し、新規ソケットを作成します。

一般的には、サブタスクが、並行サーバーから取得したクライアント ID とソケット記述子を使用してこのマクロを発行します。

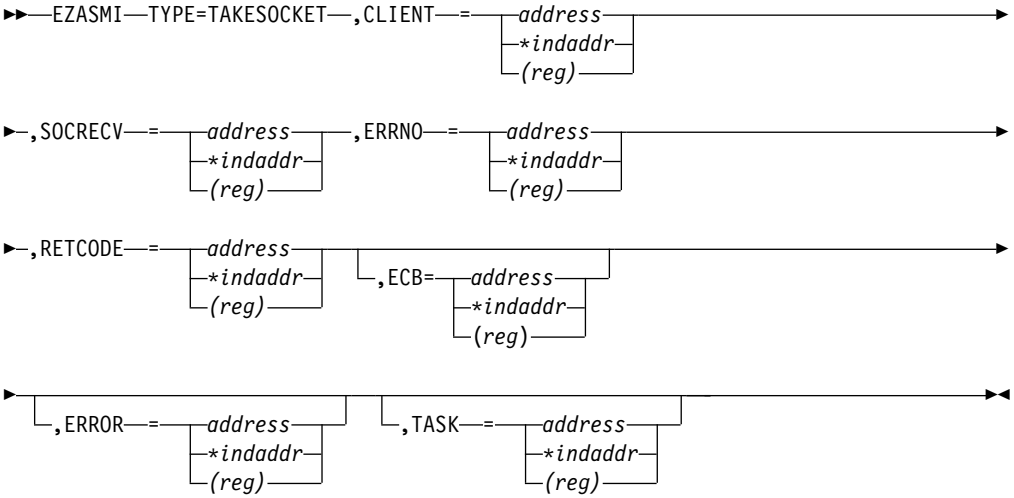
注:

1. TAKESOCKET が発行される場合、新規ソケット記述子が RETCODE で戻されます。S (ソケット記述子) パラメーターを必要とする、GETSOCKOPT などの、後で発行するマクロでこの新規ソケット記述子を使用してください。
2. 並行サーバーと反復サーバーの両方が、このインターフェースを使用します。反復サーバーは、一時点では 1 つのクライアントを扱います。並行サーバーは複数のクライアントから接続要求を受信し、それらのクライアント要求を処理する

サブタスクを作成します。サブタスクが作成されると、並行サーバーは新規ソケットを取得し、その新規ソケットをサブタスクに渡し、自身をその接続から切り離します。CICS リスナー・プログラムは、並行サーバーの 1 例です。

重要: IPv6 サポートは TCP/IP for z/VSE では使用できません。このプログラムが使用されている場合には、IPv6 アドレスまたはアドレス構造への参照は適用されません。

フォーマット



パラメーター

CLIENT

入力パラメーター。GETCLIENTID マクロで戻されるクライアント・データです。

DOMAIN

入力パラメーター。フルワード 2 進数であり、ソケットを与えるプログラムのドメインに設定されます。TCP/IP の場合、この値は、AF_INET を示す 10 進数の 2、または AF_INET6 を示す 10 進数の 19 に設定されます。

注: TAKESOCKET は同じアドレス・ファミリーのソケットを、GIVESOCKET からのみ取得できます。

NAME

8 バイトの文字フィールドであり、ソケットを与えるプログラムの VSE パーティション ID に設定されます。

TASK

入力パラメーター。8 バイトの文字フィールドを指定します。このフィールドは、GIVESOCKET 要求を発行した VSE タスクに対する、INITAPI の SUBTASK パラメーターと一致しなければなりません。

RESERVED
入力パラメーター。20 バイトの予約フィールドです。このフィールドは必須であり、内部的にのみ使用されます。

SOCRECV
入力パラメーター。ハーフワード 2 進数フィールドであり、GIVESOCKET を呼び出したアプリケーションによって割り当てられたソケット記述子の値を含みます。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
>0	3 つの戻りマスク内のすべての作動可能ソケットの数を示します。
=0	SELECT 制限時間が満了になったことを示します。
-1	ERRNO のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

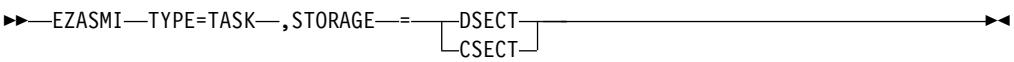
TASK

TASK マクロは、1 つのタスク内のすべてのソケット・ユーザーがアドレッシングできるタスク・ストレージ域を割り振ります。

1 つのタスク内で複数のモジュールがソケットを使用する場合、それぞれのモジュールにタスク・ストレージ・アドレスを提供する必要があります。これらのモジュールは、STORAGE=DSECT を指定した EZASMI TYPE=TASK 命令を使用して、ストレージ・マッピングを定義するべきです。

このマクロが指定されない場合、デフォルト名 EZASMTIE がストレージ・マッピングに使用されます。

フォーマット



パラメーター

STORAGE
入力パラメーター。以下のいずれかのストレージ定義を規定します。

DSECT
DSECT を生成します。

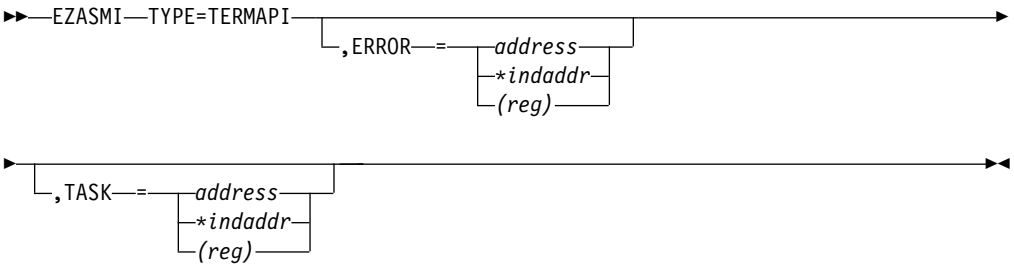
CSECT
CSECT 内で、または、より大きな DSECT の一部として使用できる、インライン・ストレージ定義を生成します。

TERMAPI

TERMAPI マクロは、INITAPI マクロで作成されたセッションを終了します。

注: INITAPI マクロと TERMAPI マクロは、同じタスクで発行されなければなりません。

フォーマット



パラメーター

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

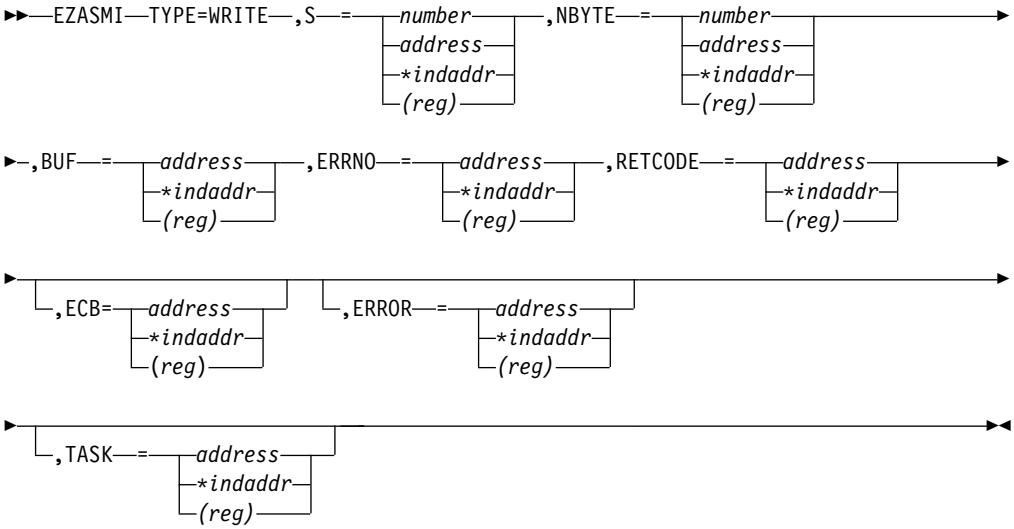
WRITE

WRITE マクロは、接続済みソケットでデータを書き込みます。WRITE マクロは、SEND マクロと似ています。

データグラム・ソケットの場合、このマクロはデータグラム全体が 1 つの TCP/IP バッファーに収まる場合は、データグラム全体を書き込みます。

ストリーム・ソケットの場合、データは、データを分離する境界のない情報ストリームとして処理されます。例えば、1000 バイトのデータを送信する場合、WRITE マクロの各呼び出しでは、1 バイト、10 バイト、または、1000 バイト全体を送信できます。WRITE マクロをループに入れ、全データが送信されるまで繰り返す必要があります。

フォーマット



パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、データを送信するソケットのソケット記述子を指定します。

NBYTE

入力パラメーター。フルワード 2 進数のアドレスまたは値であり、送信するデータのバイト数を指定します。

BUF 送信されるデータのアドレスです。BUF の長さは、最低でも NBYTE の値と同じでなければなりません。

ERRNO

出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

RETCODE

出力パラメーター。フルワード 2 進数フィールド。

値 説明

0 または **>0**

呼び出しは成功しました。値は、送信されたバイト数に設定されます。

-1 **ERRNO** のエラー・コードをチェックしてください。

ECB 入力パラメーター。以下を含む 160 バイトのフィールドを指します。

- マクロ完了時に TCP/IP によってポストされる、4 バイトの ECB。
- 状態情報を保管するのにインターフェースが使用する、156 バイトのストレージ・フィールド。

注**:** このストレージは、マクロ関数が完了し、ECB がポストされるまでは、変更されてはなりません。

ERROR

入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK

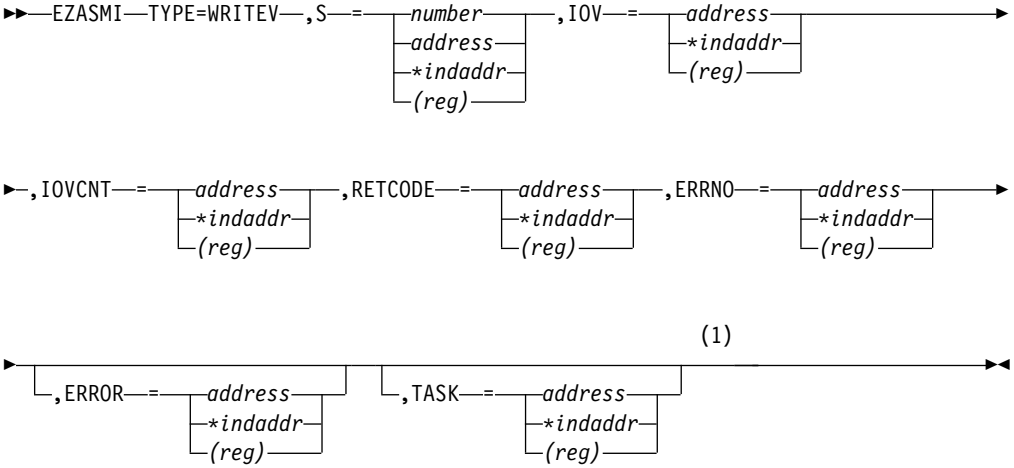
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

このマクロは、最大で NBYTE バイトのデータを書き込みます。送信されるソケット・データ用に十分なバッファー・スペースがなく、ソケットがブロッキング・モードの場合、WRITE は、追加バッファー・スペースが使用可能になるまで呼び出し元をブロックします。ソケットが非ブロッキング・モードの場合、WRITE は -1 を戻し、ERRNO を EWOULDBLOCK に設定します。非ブロッキング・モードの設定方法の説明については、 349 ページの『FCNTL』または 399 ページの『IOCTL』を参照してください。

WRITEV

WRITEV マクロは、バッファー・セットからソケットにデータを書き込みます。

フォーマット



注:

1 非同期処理用の ECB パラメーターは、この呼び出しではサポートされません (z/OS とは異なります)。

パラメーター

S 入力パラメーター。ハーフワード 2 進数のアドレスまたは値であり、データを送信するソケットを指定します。

IOV 入力パラメーター。**IOVCNT** の値と等しい数の構造体の 3 つのフルワード構造の配列。

構造体の形式は、次のとおりです。

- フルワード 1: データ・バッファーのアドレス
- フルワード 2: 予約済み
- フルワード 3: フルワード 1 で参照されるデータ・バッファーの長さ

IOVCNT
入力パラメーター。フルワードの 2 進数フィールドであり、この呼び出しに提供されるデータ・バッファー数を指定します。最大は 120 です。

RETCODE
出力パラメーター。フルワード 2 進数フィールドであり、以下のいずれかの値を戻します。

値	説明
>=0	送信されたバイト数。
-1	エラーが発生しました。ERRNO のエラー・コードをチェックしてください。

ERRNO
出力パラメーター。フルワード 2 進数フィールド。RETCODE が負の場合、ERRNO には有効なエラー番号が入っています。そうでない場合には、ERRNO は無視できます。

ERRNO 戻りコードについては、 83 ページの『ERRNO 値』 を参照してください。

ERROR
入力パラメーター。アプリケーション・プログラミング・インターフェース (API) 処理モジュールをロードできないときに制御を受け取る、プログラム内の位置。

TASK
入力パラメーター。プログラム内のタスク・ストレージ域の位置。

第 4 部 Linux へのファスト・パスの使用

第 14 章 Linux Fast Path を使用した z/VSE の実行

このセクションでは、*Linux on z Systems* へのファスト・パス 機能 (単に *Linux Fast Path* または LFP と呼ばれることもある) について説明します。

Linux Fast Path により、選択した TCP/IP アプリケーションは、z/VSE 上の TCP/IP スタックを使用せずに、Linux on z Systems 上の TCP/IP スタックと通信できます。

Linux Fast Path は、z/VM-to-z/VM、LPAR-to-LPAR、または z/VM-to-LPAR のいずれかの環境で実行できます。

- LFP を z/VM-to-z/VM環境で実行する場合、z/VSE と Linux on z Systems はいずれも同じ z/VM モードの LPAR で実行されます。 z/VSE と Linux on z Systems の間で IUCV 接続が使用されます。自分で Linux on z Systems システムを実行したくなければ、 479 ページの『第 15 章 z/VSE - z/VM IP アシスト』に説明のある z/VSE - z/VM IP Assist 機能が代替となる場合があります。
- LFP を LPAR-to-LPAR 環境で実行する場合、z/VSE と Linux on z Systems はいずれも専用の LPAR で実行されます。 z/VSE と Linux on z Systems の間で HiperSockets 接続が使用されます。自分で Linux on z Systems システムを実行したくなければ、 485 ページの『第 16 章 z/VSE Network Appliance』に説明のある z/VSE Network Appliance 機能が代替となる場合があります。
- Linux Fast Path (LFP) を z/VM-to-LPAR 環境で実行する場合、z/VSE は z/VM ゲストとして実行され、Linux on z Systems は専用の LPAR で実行されます。 HiperSockets 接続が z/VSE と Linux on z Systems 間で使用されます。この場合、z/VM ゲストの「HiperSockets 完了キュー」機能を有効にするために、z/VM バージョン 6 リリース 3 以降が必要です。自分で Linux on z Systems システムを実行したくなければ、 485 ページの『第 16 章 z/VSE Network Appliance』に説明のある z/VSE Network Appliance 機能が代替となる場合があります。

Linux on z Systems へのファスト・パス の一般的な概要については、「*IBM z/VSE 計画*」を参照してください。

LFP を使用し、LE/C で作成されたソケット・アプリケーションを使用する場合は、LE/C TCP/IP ソケット API マルチプレクサーを構成する必要があります。LE/C ソケット API マルチプレクサーを使用して、アプリケーションで使用する TCP/IP および SSL の実装を選択する方法については、 95 ページの『第 10 章 使用する TCP/IP と SSL の実装を選択』を参照してください。

このセクションでは、以下のメイン・トピックを扱います。

Linux Fast Path の概要

LFP は、すべてのソケット要求を、LFP デーモン (lfpd) が実行されている必要がある Linux on z Systems システムに透過的に転送します。このデーモンは、すべてのソケット要求を Linux TCP/IP スタックに転送することでそれらに対応します。

複数の LFP インスタンスを開始できます。

- 各インスタンスは、その ID (00 から 99) により識別され、Linux システム上の LFP デーモンへの接続を表します。
- ID は、TCP/IP スタックに使用する ID 値と対応します (例えば、`// EXEC IPNET,PARM='ID=nn'`)。
- ID はすべての TCP/IP スタックおよび LFP インスタンスの間で固有でなければなりません。したがって、LFP インスタンスと同じ ID を持つ TCP/IP スタックを持つことはできません。

以下の API を Linux Fast Path で使用できます。

- 代替 $EDCTCPV.PHASE (IJBLFPLE) からの LE/C ソケット API。
- 代替 EZA インターフェース・フェーズ IJBLFPEZ からの EZA SOCKET および EZASMI インターフェース。
- SOCKET マクロからの CSI の (*Connectivity Systems, Incorporated*) アセンブラー・ソケット・インターフェース。

注: LE/C ソケット API マルチプレクサーを使用して、アプリケーションで使用する TCP/IP および SSL の実装を選択する方法については、95 ページの『第 10 章 使用する TCP/IP と SSL の実装を選択』を参照してください。

LFP は、他のリモート・サーバーとの通信に使用される、z/VSE 上の既存の TCP/IP スタックとの置き換えは意図していません。z/VSE 上で実行するプログラム用の TCP/IP ソケット API を提供しているのは、LFP だけです。

- これらの API は、既存の API と互換性があり、未変更の既存のソケット・プログラムを LFP を使用して実行できます。
- 基本ソケット API 以外のツールは提供されていません。
- FTP サーバー/デーモン、TELNET サーバー、LPR/LPD などを実行するには、z/VSE 上に TCP/IP スタックが引き続き必要です。

Linux Fast Path の使用前提条件

z/VM-to-z/VM環境で LFP を使用するための前提条件は以下のとおりです。

- z/VM モード LPAR を使用している場合、z/VM バージョン 5 リリース 4 以降。あるいは、z/VSE によりサポートされる任意の z/VM リリース。
- https://www-03.ibm.com/systems/z/os/zvse/products/connectors.html#lfp にリストされている Linux on z Systems オペレーティング・システムのいずれか。
- IUCV 接続を使用するため、z/VSE と Linux on z Systems を、同じ z/VM 内の z/VM ゲストとして構成する必要があります。
- 両方の z/VM ゲスト (z/VSE および Linux on z Systems) で、IUCV (「ユーザー間通信機能」) を構成して有効にする必要があります。

LPAR-to-LPAR 環境または z/VM-to-LPAR 環境で LFP を使用するための前提条件は以下のとおりです。

- z/VSE および Linux on z Systems で使用する HiperSockets デバイス
  - 同一 IQD CHPID 上になければならず、必要な MTU に合わせて IOCDS で CHPARM 値 00、40、80、または C0 を使用して定義する必要があります。 *'IEDN'* および *'External Bridge'* の HiperSockets デバイスはサポートされていません。
  - レイヤー 3 デバイスとして構成しなければなりません。
- https://www-03.ibm.com/systems/z/os/zvse/products/connectors.html#lfp にリストされている Linux on z Systems オペレーティング・システムのいずれか。
- LPAR-to-LPAR
  - LPAR モードで実行されている 1 つの z/VSE システムと 1 つの Linux on z Systems システム。
- z/VM-to-LPAR
  - z/VM バージョン 6 リリース 3 以降でゲストとして実行されている z/VSE、および LPAR モードで実行されている Linux on z Systems。

    注: z/VM ゲストが、仮想 HiperSockets デバイスではなく実 HiperSockets デバイスを (前述のように) 使用するようにしてください。

Linux Fast Path の使用時の制限

以下に示すのは、LFP の使用時の制限です。

- CSI (*Connectivity Systems, Incorporated*) インターフェースである SOCKET マクロの場合、LFP は以下の接続タイプのみをサポートします。
  - TCP
  - UDP
  - CONTROL

  その他の接続タイプ (CLIENT、TELNET、FTP、RAW など) はサポートされず、LFP で使用した場合は拒否されます。
- CONTROL タイプの接続の場合、以下のコマンドのみがサポートされます。
  - GETHOSTBYNAME

- GETHOSTBYADDR
- GETHOSTNAME
- GETHOSTID

詳しくは、「*TCP/IP for VSE Programmer's Guide*」の個々のマクロの説明を参照してください。

- CONTROL タイプの接続の場合、以下のコマンド (Barnard Software, Incorporated 提供) もサポートされます。
  - NTOP
  - PTON
  - GETVENDORINFO

  詳しくは、Barnard Software, Incorporated 発行の、「*IPv6/VSE Programming Guide*」を参照してください。
- SSL API を (`gsk_nnn` 機能を用いて) 使用する場合、異なる ID を使用して TCP/IP for z/VSE TCP/IP スタックを稼働させる必要があります。通信フローが LFP を使用するとしても、SSL 機能は TCP/IP スタックからのサービスを使用します。

Linux Fast Path を使用しない場合の通信フロー

このトピックは、Linux Fast Path を使用しない一般構成を詳細に説明しています。 449 ページの図 22 にこれを示します。この構成では、z/VSE の下で実行する Db2 クライアントは、Linux の下で実行する Db2 Server for Linux と通信します。

この構成は、Linux on z Systems 上で実行する z/VSE 提供の他のプログラムを実行する場合にも適用されます (VSE Redirector Server、Redirector Handler など)。 449 ページの図 22 は、次のことを示しています。

- z/VSE と Linux on z Systems との間の通信は、HiperSockets を使用して確立されている。
- すべての通信は、z/VSE および Linux on z Systems 上の TCP/IP スタック経由でルーティングする必要がある。

高速の HiperSockets 接続では、TCP 処理および IP 処理で必要とされるオーバーヘッドにより、システム使用率が容易に高くなる可能性があります。

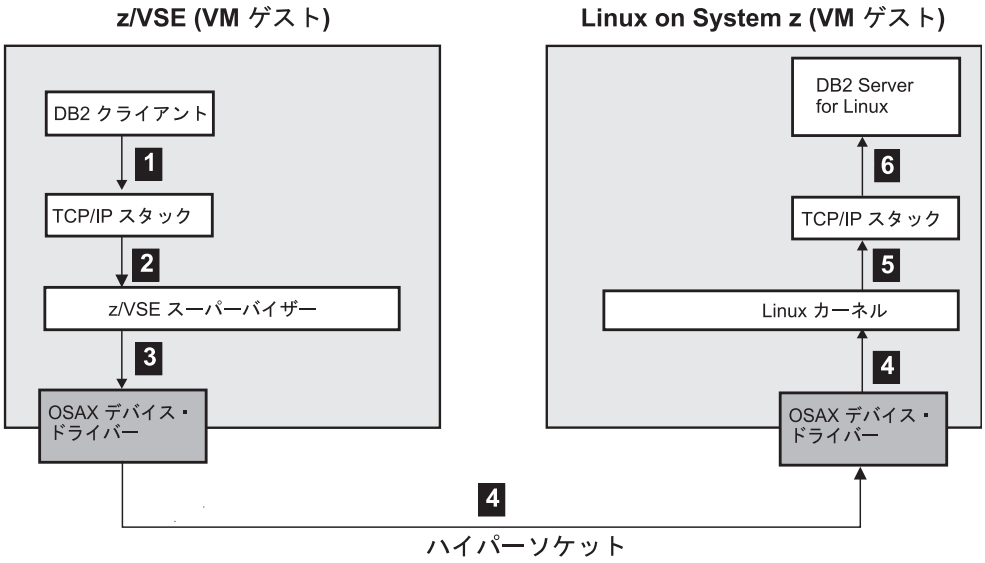


図 *22. HiperSockets* からの *VM* ゲスト間の通信

図 22 で示す処理は、以下のとおりです。

1 データは、z/VSE アプリケーション (この例では Db2 クライアント) から、別のパーティションで実行している TCP/IP スタックに渡されます。これはパーティション間通信メカニズムを使用して実行され、z/VSE スーパーバイザーでのいくつかのディスパッチング・アクティビティーが関係しています。

2 TCP/IP は、1 つまたは複数の TCP パケットを、ユーザー・アプリケーションからのデータで作成します。これにより TCP ヘッダーと IP ヘッダーが作成されます。この処理には、再送信処理、シーケンス番号と確認応答、チェックサムの計算などが含まれます。

3 TCP/IP スタックは、HiperSockets で使用するために、パケットをネットワーク・デバイス・ドライバーに渡します (例えば、OSAX デバイス・ドライバー)。

4 HiperSockets ネットワークはパケットを Linux イメージに転送します。

5 Linux HiperSockets デバイス・ドライバーはパケットを受け取り、それを TCP/IP スタックに渡します。Linux 上の TCP/IP スタックは、IP および TCP ヘッダーを検査してアンパックします。この処理には、再送信処理、シーケンス番号と確認応答、チェックサムの妥当性検査などが含まれます。

6 TCP/IP スタックは、データを Linux 上で実行しているアプリケーション (この例では DB2® サーバー) に渡します。Db2 サーバーはデータを受け取って処理します。

Linux 上の Db2 サーバーから z/VSE 上の Db2 クライアントに返されるデータの場合、上記の 6 つの同じステップが逆順で実行されます。

z/VM-to-z/VM環境で Linux Fast Path を使用した場合の通信フロー

このトピックでは、z/VM-to-z/VM環境で Linux Fast Path を使用する典型的な構成について詳細に説明します。 図 23 にこれを示します。この構成では、z/VSE の下で実行する DB2 クライアントが、Linux の下で実行する Db2 Server for Linux と通信する例を使用しています。

図 23 は、次のことを示しています。

- z/VSE と Linux on z Systems との間の通信は、*IUCV* 通信パスを使用して確立されている。
- Linux Fast Path は、z/VSE と Linux on z Systems との間の IUCV 通信パスを、以下のものを使用せずに 確立します。
  - z/VSE 上の TCP/IP スタック
  - OSA Express アダプター

この例で説明されている処理の流れは、Linux 上で実行される他の z/VSE 提供プログラム (VSE Redirector Server、VSE Script Server、Redirector Handler など) の実行時にも当てはまります。

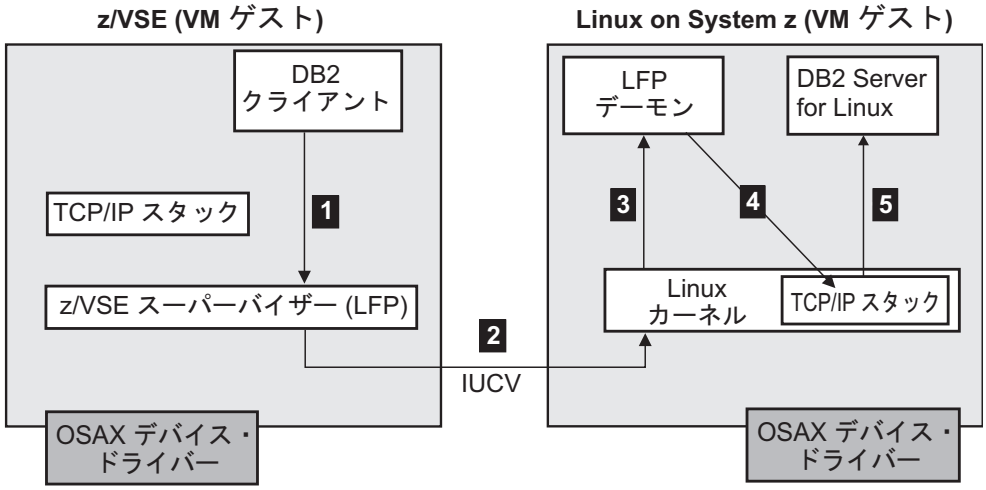


図 *23. Linux Fast Path* からの *VM* ゲスト間の通信

図 23 で示す処理は、以下のとおりです。

1 送信されるデータは、z/VSE 上で実行する LFP に渡されます。

2 LFP は IUCV チャネルを介してデータを Linux イメージに送信します。

3 Linux IUCV デバイス・ドライバーはデータを受け取り、それを Linux イメージ上で実行されている lfpd に渡します。その後、`lfpd` は、データを処理し、それをソケット呼び出しに変換します。

4 ソケット呼び出しは、TCP/IP スタックにより処理されます。データは同じ Linux システム上で実行するアプリケーションに送信されるので (この例では DB2 サーバー)、TCP/IP スタックは単にそのデータを DB2 サーバーに直接転送します。TCP/IP は、 449 ページの図 22 で必要であったプロセス集約的なステップの実行には必要ありません。

5 DB2 サーバーはデータを受け取って処理します。

Linux 上の DB2 サーバーから z/VSE 上の DB2 クライアントに返されるデータの場合、上記の 5 つの同じステップが逆順 で実行されます。

LPAR-to-LPAR 環境または z/VM-to-LPAR 環境での Linux Fast Path 使用時の通信フロー

このトピックでは、LPAR-to-LPAR 環境または z/VM-to-LPAR 環境で Linux Fast Path を使用する、典型的な構成について詳細に説明します。 図 24 にこれを示します。この構成では、z/VSE の下で実行する Db2 クライアントが、Linux の下で実行する Db2 Server for Linux と通信する例を使用しています。

図 24 は、次のことを示しています。

- z/VSE と Linux on z Systems との間の通信は、HiperSockets 通信パスを使用して確立されている。
- LFP は、z/VSE と Linux on z Systems との間の HiperSockets 接続を、以下のものを使用せずに 確立している。
  - z/VSE 上の TCP/IP スタック
  - OSA Express アダプター

LPAR-to-LPAR 環境および z/VM-to-LPAR 環境では、LFP は HiperSockets 接続を使用します。 この例で説明されている処理の流れは、Linux 上で実行される他の z/VSE 提供プログラム (VSE Redirector Server、VSE Script Server、Redirector Handler など) の実行時にも当てはまります。

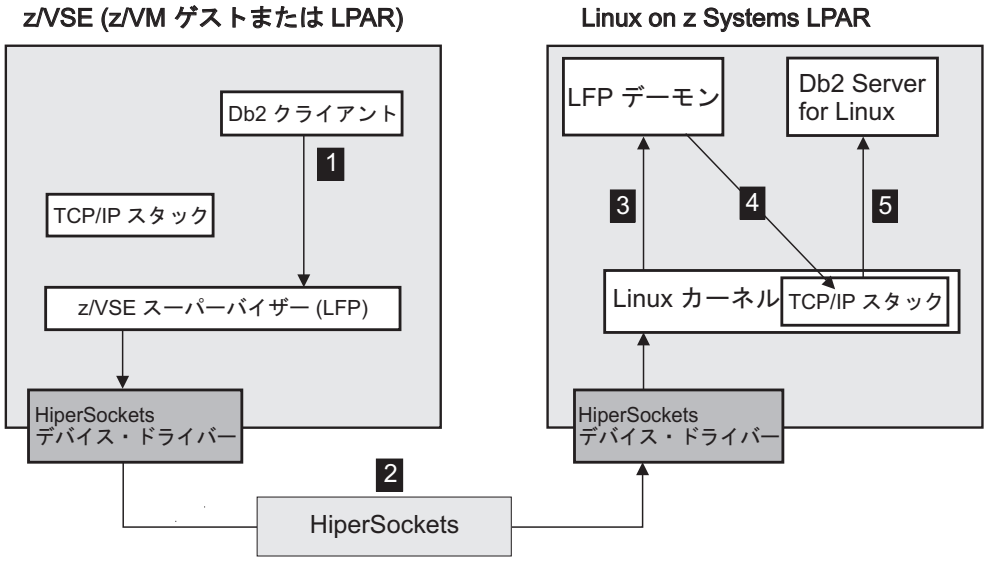


図 *24. HiperSockets* からの *LPAR* 間の通信

図 24 で示す処理は、以下のとおりです。

1 送信されるデータは、z/VSE 上で実行する LFP に渡されます。

2 LFP は HiperSockets デバイス・ドライバーを呼び出して、データを Linux イメージに送信します。HiperSockets 完了キュー 機能は、データ伝送が正常に終了することを保証します。

3 Linux デバイス・ドライバーはデータを受け取り、それを Linux イメージ上で実行されている lfpd に渡します。その後、lfpd は、受け取ったデータを処理し、それをソケット呼び出しに変換します。

4 ソケット呼び出しは、TCP/IP スタックにより処理されます。データは同じ Linux システム上で実行するアプリケーションに送信されるので (この例では Db2 サーバー)、TCP/IP スタックは単にそのデータを Db2 サーバーに直接転送します。TCP/IP は、 449 ページの図 22 で必要であったプロセス集約的なステップの実行には必要ありません。

5 Db2 サーバーはデータを受け取って処理します。

Linux 上の Db2 サーバーから z/VSE 上の Db2 クライアントに返されるデータの場合、上記の 5 つの同じステップが逆順 で実行されます。

Linux Fast Path を使用するための Linux on z Systems の準備

Linux on z Systems 上で実行する LFP デーモン (*lfpd*) を事前にインストールしておく必要がありますが、そのためには、user root としてログオンする必要があります。

LFP デーモンのコピーを取得するには、以下の 2 とおりの方法があります。

1. インターネットから
   - インターネットからクライアントを取得するには、z/VSE の Web サイト (http://www.ibm.com/systems/z/os/zvse/downloads/) にアクセスします。

     「*Linux* ファスト・パス」セクションに進み、ファイル *vselfpdnnn.zip* を一時ディレクトリーにダウンロードして、Linux ファスト・パスをインストールします。

     注: *nnn* は、現行の VSE バージョンを表します (例えば、vselfpd610.zip)。
2. Extended Base Tape から
   - クライアントを Extended Base Tape から取得するには、Extended Base Tape に収録されている VSE Connectors Workstation Code コンポーネント (5686CF8-38 CONN.C/W) をインストールします。

VSE Connectors Workstation Code コンポーネントのインストール後は、lfpd W-book ijblfplx.w は、z/VSE サブライブラリー PRD2.PROD に保管されます。

TCP/IP for z/VSE の FTP (ファイル転送プログラム) ユーティリティーを使用して、ijblfplx.w を一時ディレクトリーに以下の設定でダウンロードします。

- ijblfplx.w をバイナリー・モードでダウンロードする。
- UNIX モードがオフになっていることを確認します。そうなっていない場合、ijblfplx.w は、バイナリーを指定した場合でも ASCII モードでダウンロードされます。UNIX モードは、VSE FTP デーモンの 1 パラメーターです。一部の FTP クライアントは、UNIX モードを強制的にオンにする場合があります。

以下の例は、バッチ FTP クライアントを使用した ijblfplx.w の正常な転送を示しています。UNIX のモード設定は太字で強調表示されています。

```
linlfp:/root/vselfpd # ftp 10.0.0.1
Connected to 10.0.0.1.
220-TCP/IP for VSE Internal FTPDAEMN 02.01.03 20150428 21.14
    Copyright (c) 1995,2015 Connectivity Systems Incorporated

220 Service ready for new user.
User (10.0.0.1:(none)): sysa
331 User name okay, need password.
Password:
230 User logged in, proceed.
ftp> cd prd2
250 Requested file action okay, completed.
ftp> cd prod
250 Requested file action okay, completed.
ftp> binary
200 Command okay
ftp> get ijblfplx.w
200 Command okay.
150-File: PRD2.PROD.IJBLFPLX.W
    Type: Binary Recfm: FB Lrecl:    80 Blksize:      80
    CC=ON  UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON  NAT=NO
150 File status okay; about to open data connection
226-Bytes sent:         123,456
    Records sent:        12,345
    Transfer Seconds:        16.52 (    290K/Sec)
    File I/O Seconds:         3.94 (  1,548K/Sec)
226 Closing data connection.
123456 bytes received in 17,12 seconds (277,91 Kbytes/sec)
ftp> bye
221 Service closing control connection.
```

ダウンロードしたファイルを、以下のコマンドで ijblfplx.zip にリネームします。

```
linlfp:/root/vselfpd # mv IJBLFPLX.W ijblfplx.zip
```

zip ファイルを以下のようにして解凍します。

```
linlfp:/root/vselfpd # unzip ijblfplx.zip
```

rpm インストール・パッケージ vselfpd-1.3.0-1.s390x.rpm は、この zip ファイルから取り出します。実際のファイル名は、バージョンまたはリリースに応じて異なる場合があることに注意してください。

rpm インストール・パッケージをインストールするには、以下のようにして rpm プログラムを呼び出します。

```
linlfp:/root/vselfpd # rpm –i vselfpd-1.3.0-1.s390x.rpm
```

古いバージョンの vselfpd が既にインストールされている場合は、インストールされているパッケージを以下のようにしてアップグレードできます。

```
linlfp:/root/vselfpd # rpm –U vselfpd-1.3.0-1.s390x.rpm
```

rpm パッケージは、以下のファイルをインストールします。

使用法 ファイルの場所

lfpd バイナリー
/opt/ibm/vselfpd/sbin/lfpd

lfpd 管理ツール・バイナリー
/opt/ibm/vselfpd/sbin/lfpd-admin

lfpd 制御スクリプト

/opt/ibm/vselfpd/sbin/lfpd-ctl

lfpd SysV 開始スクリプト

/etc/init.d/vselfpd

lfpd バイナリーへのパスを確立する **user root** 用プロファイル

/etc/profile.d/vselfpd.sh

コマンド・ライン・プログラム用の **man** ページ

/usr/share/man/man8/lfpd.8.gz
/usr/share/man/man8/lfpd-admin.8.gz
/usr/share/man/man8/lfpd-ctl.8.gz

使用可能な構成、およびサンプル構成が入ったディレクトリー

/etc/opt/ibm/vselfpd/confs-available/
/etc/opt/ibm/vselfpd/confs-available/lfpd-SAMPLE.conf

使用可能になっている構成が入ったディレクトリー

/etc/opt/ibm/vselfpd/confs-enabled

注**:** ログオン・プロファイルは、PATH 環境変数を `/opt/ibm/vselfpd/sbin` ディレクトリーに設定します。`user root` だけが `lfpd` を開始できるので、PATH はそのユーザーに対してのみ設定されます。ログオン・プロファイルは、`user root` として直接ログオンした場合にのみ実行されます。システムにそれ以外のユーザーとしてログオンした場合は、`user root` にコマンド `su -` を使用して切り替えることができますが、ログオン・プロファイルは実行されず、PATH は設定されません。

LFP を HiperSockets を介して実行する場合、`lfpd` を開始するには、HiperSockets 装置の `qeth sysfs` 属性の *hsuid*/リンクを設定する必要があります (例: 0.0.8200 / hsi0)。システムは、指定された *hsuid* から IPv6 link-local アドレスを生成します。したがって、*hsuid* を HiperSockets ネットワーク全体およびシステム内で固有にする必要があります。つまり、同じ *hsuid* を使用する 2 つの HiperSockets 装置 (例: hsi0 と hsi1) を持つことはできません。

hsuid 属性値は大/小文字を区別します。LFP と共に使用するには、値をすべて大文字で指定する必要があります。HiperSockets 装置の IP アドレスまたはサブネット・マスクを指定しないでください。HiperSockets 装置は LFP 用または通常の TCP/IP ネットワーク用に使用できますが、LFP 用および通常の TCP/IP ネットワーク用として同時に使用することはできません。

`lfpd` の自動開始機能 (init.d 経由) を使用して、ブート時に構成済みの LFP インスタンスを開始する場合は、`lfpd` が開始される前に *hsuid* を設定する必要があります。`qeth sysfs` 属性 *hsuid* を構成するには、Linux 配布版で提供されているネットワーク構成ツールを使用します。sysconfig 構成ファイルの `QETH` オプション "*hsuid=nnnnnn*" を設定します。SuSE 配布版の場合は、YAST を使用して構成できます。

hsuid を手動で設定するには、以下のコマンドを使用できます (この例では、リンク名 *hsi0* と装置アドレス *8200* が使用されています)。

```
ifconfig hsi0 down
echo 0 > /sys/bus/ccwgroup/devices/0.0.8200/online
echo "SYSNAME " > /sys/bus/ccwgroup/devices/0.0.8200/hsuid
echo 1 > /sys/bus/ccwgroup/devices/0.0.8200/online
ifconfig hsi0 up
```

VM および LPAR 環境で Linux カーネルのメモリー不足の状況を回避するには、以下の値を 1MB 以上に増加する必要があります。

```
sysctl -w net.core.rmem_max=1048576
sysctl -w net.core.rmem_default=1048576
sysctl -w net.core.wmem_max=1048576
sysctl -w net.core.wmem_default=1048576
```

MTU サイズが非常に大きく、メッセージ制限 (HS_MSGLIMIT または IUCV_MSGLIMIT) が高い場合は、これらの設定をさらに増加することが必要になる場合があります。sysctl コマンドは、`lfpd` が開始される前に実行される必要があります。

z/VSE 上でソケット API と Linux Fast Path とを使用するための準備

ICCF ライブラリー 59 にあるスケルトン SKLFPACT を使用して、z/VSE システムに対して必要なすべての変更を実行します。これによりアプリケーションは LFP を使用できるようになります。SKLFPACT の一部のステップは、すべての IPL 後に繰り返す必要があります。一部のステップは、アプリケーションの実行時に JCL に追加する必要があります (例えば、SYSPARM 設定)。

LE/C ソケット API

LE/C ソケット API は、$EDCTCPV.PHASE に実装されています。以下の $EDCTCPV フェーズを、システム内に存在させることができます。

- LE ダミー・フェーズは PRD2.SCEEBASE に置かれます。
- TCP/IP for z/VSE のインターフェース・フェーズは PRD2.TCPIPC に置かれます。
- その他の TCP/IP スタックは他のライブラリーに置かれます。

LIBDEF SEARCH 文は、使用するインターフェース・フェーズを制御します。

LFP の LE/C インターフェース・フェーズは、IJSYSRS.SYSLIB 内に、IJBLFPLE.PHASE として付属しています。実行中のすべての LFP インスタンスの ID に対して、LFP LE/C TCP/IP インターフェース・フェーズ IJBLFPLE を使用するように、LE/C TCP/IP Socket API Multiplexer を構成する必要があります。マルチプレクサーを構成するには、ICCF ライブラリー 62 にあるスケルトン EDCTCPMC を使用します。

以下のコマンドを使用して、すべての LFP インスタンスに対して項目を追加できます。

```
EDCTCPME SYSID='01',PHASE='IJBLFPLE'
```

スケルトンをサブミットして、ゼロの戻りコードが戻されることを確認します。TCP/IP ソケットを CICS 下で使用する場合、マルチプレクサー構成をアクティブにするために、以下のコマンドでの再ロードが必要になることがあります。

```
CEMT SET PROG(EDCTCPMC) NEW
```

詳しくは、 101 ページの『第 11 章 TCP/IP による LE/VSE C ソケット・インターフェースのサポート』を参照してください。

注: LFP LE/C TCP/IP インターフェース・フェーズ IJBLFPLE は、LFP に対してのみ機能し、他の TCP/IP スタックでは機能しません。

EZA ソケットと EZASMI インターフェース

EZA ソケットと EZASMI インターフェースを用いて、使用するインターフェース・モジュールを指定できます。LFP に対しては、EZA インターフェース・モジュール IJBLFPEZ を使用する必要があります。

JCL パラメーター "EZA$PHA" を、LFP を使用するすべてのジョブに設定する必要があります。そうするには、以下のコマンドを使用します。

```
// SETPARM [SYSTEM,]EZA$PHA=IJBLFPEZ
```

EZA ソケットまたは EZASMI インターフェースを CICS 下で使用する場合は、EZA 'TASK-RELATED-USER-EXIT' (TRUE) をアクティブにする必要があります。詳しくは、 93 ページの『EZA インターフェースに関する CICS の考慮事項』を参照してください。

インスタンス ID の指定

表 10 は、インスタンス ID の指定方法、およびその適用先について示しています。設定は、表にリストされている順序で、上から下に検査されます。

表 *10.* インスタンス *ID* の指定

	LE/C API	**EZA API**	**CSI SOCKET** マクロ
'LFP$ID' (環境変数)	x		
// SETPARM [SYSTEM,]LFP$ID=NN	x	x	
'SYSID' (環境変数)	x		
INITAPI に渡される IDENT.TCPNAME 呼び出し		x	
SOCKET マクロの ID パラメーター			x
// OPTION SYSPARM='NN'	x	x	x
デフォルト '00'	x	x	x

ID を指定しない場合は、'00' の ID が使用されます。

注: SSL 関数 (gsk_*) を使用する場合、ネットワーク通信に LFP を使用している場合でも、CSI TCP/IP スタックを実行しておく必要があります。この場合、CSI TCP/IP スタックの ID は、`// OPTION SYSPARM` 文または `// SETPARM [SYSTEM,]LFP$SSL=NN` 文で指定する必要があります。LFP インスタンスの ID は、異なるものでなければなりません。

CICS タスク分離オプション

LFP は、CICS タスクを相互に分離します。これはつまり、ある CICS タスクにより割り振られたソケットは、そのソケットが GIVESOCKET/TAKESOCKET 呼び出しにより他の CICS タスクに渡される場合を除き、別の CICS タスクでは使用できないという意味です。ただし一部のプログラムは、GIVESOCKET/TAKESOCKET を使用せずに、ソケットを CICS タスク間で受け渡す処理に依存しています。このようなプログラムが LFP で機能するように、そのプログラムに対して以下の JCL ステートメントを指定する必要があります。

```
// SETPARM [SYSTEM,]LFP$CIC=SHARE
```

この設定は、LE/C ソケット・インターフェースおよび EZA インターフェースに適用されます。

例えば、Db2 (クライアントまたはサーバー) アプリケーション・リクエスターは、CICS 下で実行している場合は、ソケット共有を必要とします。

Linux Fast Path の構成

LFP を使用可能にするには、z/VM、Linux on z Systems、および z/VSE でいくつかの構成パラメーターを設定する必要があります。続くいくつかのセクションの説明と同じ順序で、構成をセットアップして使用可能にします。

注: 構成パラメーターの説明で 10 進数が必要であると記述されている場合、特に断りがなければその数値はバイト数を示します。

z/VM

z/VM-to-z/VM 環境では、LFP は IUCV を基本の通信手段として使用します。z/VSE および Linux on z Systems ゲストは IUCV 用に構成する必要があります。

注: これは z/VM-to-LPAR 環境にはあてはまりません。この環境では LFP は HiperSockets 接続を使用します。

パラメーターの詳細については、z/VM の資料を参照してください。

ゲスト・システムの以下の z/VM パラメーターが関係あります。

IUCV ALLOW
他のすべての仮想マシンに、この仮想マシンとの通信パスの確立を許可します。通信を開始する仮想マシンには、これ以外の許可は必要ありません。

IUCV ANY
z/VM ゲスト仮想マシンに、他のすべての z/VM ゲスト仮想マシンとの通信パスの確立を許可します。

IUCV MSGLIMIT
MSGLIMIT は、許可されている任意のパス上での、許可されている未解決メッセージの最大数を定義します。値は 16 から 65535 の範囲内でなければなりません。MSGLIMIT が z/VM ゲスト仮想マシン構成および z/VSE LFP 構成に指定された場合、その両方の最低値が IUCV 通信パスに適用されます。

このパラメーターはオプションです。

OPTION MAXCONN *maxno*
この仮想マシンに許可されている IUCV 接続の最大数を指定します。MAXCONN オペランドが省略された場合、デフォルトは 64 です。最大は 65535 です。

注: z/VSE は、最大 10 の並列 IUCV 接続に対応できます。

Linux on z Systems

各 LFP デーモン (lfpd) は独自の構成を持っています。これは lfpd の開始時に読み取られ、lfpd の実行中は変更できません。構成はファイルで提供する必要があります。

注: lfpd 制御スクリプトの lfpd-ctl には、構成ファイル名の命名規則が必要です。各構成ファイルは lfpd-XXX.conf の形式で命名する必要があります。ここで XXX は、構成ファイルに指定される IUCV_SRC_APPNAME または HS_SRC_APPNAME です。ファイル名の XXX 文字は、大文字で指定する必要があります。例えば、LINR02 という IUCV_SRC_APPNAME を持つ構成ファイルは、lfpd-LINR02.conf と名付けられます。

使用可能なすべての構成ファイルは、/etc/opt/ibm/vselfpd/confs-available ディレクトリーに保管する必要があります。/etc/opt/ibm/vselfpd/confs-enabled ディレクトリーには、有効になっている構成ファイルが入ります。構成が有効になっている場合は、lfpd 制御スクリプトの lfpd-ctl を使用して簡単に開始できます。さらに、SysV 開始スクリプトにより、Linux システムの始動時に有効なすべての構成が開始します。有効にする、使用可能な各構成に対して、シンボリック・リンクを作成する必要があります。

例えば、LINR02 という名前で構成を作成した場合、構成ファイルは /etc/opt/ibm/vselfpd/confs-available/lfpd-LINR02.conf となります。この構成を有効にすることを計画したとします。構成ファイルから有効にする構成ディレクトリーへのシンボリック・リンクを配置するには、以下のコマンドを使用します。

```
ln -s /etc/opt/ibm/vselfpd/confs-available/lfpd-LINR02.conf
 /etc/opt/ibm/vselfpd/confs-enabled
```

構成ファイル内の各パラメーターは、key = value の形式で指定する必要があります。構成内の空の行は無視されます。行の先頭の文字「#」はコメント行を示しますが、これも無視されます。構成パラメーターは大/小文字を区別しません。

IUCV 固有の構成
これらのパラメーターは **IUCV** 接続でのみ使用されます。

IUCV_SRC_APPNAME
IUCV_SRC_APPNAME の値は、1 から 8 文字の英数字で指定する必要があります。この値はローカル・ポートを定義し、Linux on z Systems システムで固有である必要があります。このパラメーター値は、z/VSE 上の LFP インスタンスに使用される IucvDestAppName と一致している必要があります。このパラメーターは必須です。

PEER_IUCV_APPNAME
PEER_IUCV_APPNAME の値は、1 から 8 文字の英数字で指定する必要が

あります。この値は、z/VSE 上のインスタンスの IUCV アプリケーション名 (IucvSrcAppName) を定義します。このパラメーターを設定すると、lfpd はすべての着信 IUCV 接続のソース・アプリケーションの名前を検査します。名前が構成内の名前と一致しない場合、着信接続は取り消されます。PEER_IUCV_APPNAME パラメーターを構成内に指定しない場合、着信 IUCV 接続のソース・アプリケーションの名前は検査されません。このパラメーターはオプションです。

PEER_IUCV_VMID

PEER_IUCV_VMID の値は、1 から 8 文字の英数字で指定する必要があります。この値は、z/VM ゲスト・システムの名前を定義します。このパラメーターを設定すると、lfpd はすべての着信 IUCV 接続のソース z/VM ゲストの名前を検査します。名前が構成内の名前と一致しない場合、着信接続は取り消されます。PEER_IUCV_VMID パラメーターを構成内に指定しない場合、着信 IUCV 接続のソース z/VM ゲストの名前は検査されません。このパラメーターはオプションです。

IUCV_MSGLIMIT

IUCV_MSGLIMIT の値は 16 から 65535 の 10 進数でなければなりません。IUCV_MSGLIMIT 値は、z/VSE 上のインスタンスによってまだ受け取られていない未解決 IUCV メッセージの最大数を指定します。推奨値は 1024 です。この値を高くするとパフォーマンスが向上する可能性がありますが、z/VSE 上でより多くのメモリーが必要になります。このパラメーターは必須です。

IUCV_MTU_SIZE|MTU_SIZE

IUCV_MTU_SIZE (MTU_SIZE も受け入れ可能) の値は 256 から 65535 (64 KB) の 10 進数でなければなりません。IUC_MTU_SIZE は、転送されるパケットの最大サイズを指定します。推奨値は 8192 (8 KB) です。この値を高くするとパフォーマンスが向上する可能性がありますが、さらに多くのメモリーが必要になります。IUCV_MTU_SIZE 値は、z/VSE 上のインスタンスに指定した IucvMTU 値と正確に一致しなければなりません。このパラメーターは必須です。

注**:** z/VSE VIA を使用する場合、MTU_SIZE を指定する必要があります。IUCV_MTU_SIZE は受け入れられません。

HiperSockets 固有の構成

これらのパラメーターは **HiperSockets** 接続でのみ使用されます。

HS_MSGLIMIT

HS_MSGLIMIT の値は 1 から 128 の 10 進数でなければなりません。デフォルト値は 128 です。HS_MSGLIMIT は、lfpd によってまだ受け取られていない未解決メッセージの最大数を指定します。この値を高くするとパフォーマンスは向上する可能性がありますが、さらに多くのメモリーが必要になります。このパラメーターはオプションです。

専用の HiperSockets ネットワークでは、この値は、HiperSockets 装置に対して構成された buffer_count の値に一致する必要があります。buffer_count のデフォルト値も 128 です。このパラメーターはオプションです。

HS_SRC_APPNAME

HS_SRC_APPNAME の値は、1 から 8 文字の英数字で指定する必要があります。この値はローカル・ポートを定義し、Linux on z Systems システムで固有である必要があります。このパラメーター値は、z/VSE 上の LFP インスタンスに使用される HSDestAppName と一致している必要があります。このパラメーターは必須です。

HS_SRC_SYSTEMNAME

HS_SRC_SYSTEMNAME の値は、1 から 8 文字の英数字で指定する必要があります。この値は、`lfpd` が使用する HiperSockets 装置の *hsuid* を定義します。内部的に、*hsiud* は、HiperSockets 装置の初期化に使用される IPv6 link-local アドレスに変換されます。複数の `lfpd` を同一 HiperSockets 装置上で実行できますが、この場合、各 lfpd で異なる HS_SRC_APPNAME を使用する必要があります。このパラメーター値は、z/VSE 上の LFP インスタンスに使用される HsDestSystemName と一致している必要があります。このパラメーターは必須です。

PEER_HS_APPNAME

PEER_HS_APPNAME の値は、1 から 8 文字の英数字で指定する必要があります。この値は、z/VSE 上のインスタンスの HsSrcAppName を定義します。このパラメーターを設定すると、`lfpd` はすべての着信接続のソース・アプリケーションの名前を検査します。名前が構成内の名前と一致しない場合、着信接続は取り消されます。PEER_HS_APPNAME を構成内に指定しない場合、着信接続のソース・アプリケーションの名前は検査されません。このパラメーターはオプションです。

PEER_HS_SYSTEMNAME

PEER_HS_SYSTEMNAME の値は、1 から 8 文字の英数字で指定する必要があります。この値は、z/VSE 上のインスタンスの HsSrcSystemName を定義します。このパラメーターを設定すると、`lfpd` はすべての着信接続のソース・システムの名前を検査します。名前が構成内の名前と一致しない場合、着信接続は取り消されます。PEER_HS_SYSTEMNAME を構成内に指定しない場合、着信接続のソース・システムの名前は検査されません。このパラメーターはオプションです。

共通の構成

これらのパラメーターは、**IUCV** 接続と **HiperSockets** 接続の両方で使用できます。

DEVICETYPE

DEVICETYPE の値は *IUCV* または *HS* のいずれかでなければなりません。IUCV が指定された場合、転送メカニズムとして IUCV が使用されます。HS は、HiperSockets を使用する必要があることを指定します。*IUCV* はデフォルトです。このパラメーターはオプションです。

注**:** z/VSE VIA を使用する場合、このパラメーターを省略する必要があります。z/VSE VIA は、転送メカニズムとして IUCV を常に使用し、DEVICETYPE パラメーターを受け入れません。

INITIAL_IO_BUFS

INITIAL_IO_BUFS の値は 10 進数でなければなりません。
INITIAL_IO_BUFS 値は、`lfpd` のスタートアップ時に割り振られる入出力

バッファーの数を指定します。Lfpd は、データの送受信のために固定サイズのバッファーを割り振ります。バッファーのサイズは主に、z/VSE 上のインスタンスへの接続の MTU サイズに依存します。実行時には、lfpd は必要な場合にさらに多くのバッファーを自動的に割り振ります。推奨値は 128 です。このパラメーターはオプションです。

MAX_SOCKETS

MAX_SOCKETS の値は 10 進数でなければなりません。MAX_SOCKETS 値は、lfpd が開くことができるソケットの最大数を指定します。Linux システム上の通常のユーザーは、一般にはプログラム当たり最大 1024 のソケットを開くことが許可されます。この値は lfpd には低すぎる場合があります。したがって、MAX_SOCKETS 値はさらに高い値に設定できます。root ユーザーで開始された Lfpd は、最大ソケット制限を、指定された値に設定します。パフォーマンス上の理由から、MAX_SOCKETS は 256 の倍数にする必要があります。デフォルト値は 1024 です。このパラメーターはオプションです。

MAX_VSE_TASKS

MAX_VSE_TASKS パラメーターの値は 10 進数でなければなりません。MAX_VSE_TASKS 値は、z/VSE 上で実行するタスクの最大数を指定します。この値は、z/VSE システムのセットアップに応じて異なります。デフォルトは、512 です。このパラメーターはオプションです。

WINDOW_SIZE

WINDOW_SIZE の値は 10 進数でなければなりません。この値は、ソケットの lfpd 受信ウィンドウのサイズを指定します。lfpd 受信ウィンドウは、lfpd が実際にデータを外部ソケットに転送することなしに、lfpd によって、z/VSE 上のインスタンスから受け取ることができるデータの量を定義します。この値を高くするとパフォーマンスは向上する可能性がありますが、さらに多くのメモリーが必要になります。最小値は 4096 (4 KB) です。推奨値は 65535 です。

WINDOW_THRESHOLD

WINDOW_THRESHOLD の値は、パーセンテージを示す 1 から 100 の 10 進数でなければなりません。WINDOW_THRESHOLD は、lfpd が通知を LFP インスタンスに送信する前に WINDOW_SIZE が達する必要があるパーセンテージを指定します。この値を高くするとパフォーマンスは向上する可能性がありますが、さらに多くのメモリーが必要になります。推奨値は 25 です。

VSE_CODEPAGE

VSE_CODEPAGE の値は、Linux システム上で iconv に認識されるコード・ページを指定するストリングでなければなりません。テキスト・データを受け取ったり返したりする関数の場合 (gethostbyname() など)、lfpd は EBCDIC (z/VSE) <-> ASCII (Linux) 変換を実行する必要があります。lfpd は、指定された VSE_CODEPAGE を EBCDIC コード・ページとして使用します。指定可能な値は EBCDIC-US です。

LINUX_CODEPAGE

LINUX_CODEPAGE の値は、Linux システム上で iconv に認識されるコ

ード・ページを指定するストリングでなければなりません。このコード・ページは、VSE_CODEPAGE と対を成しています。デフォルトは 850 です。このパラメーターはオプションです。

SUPPORT_GETXXXENT
SUPPORT_GETXXXENT の値は *yes* または *no* のいずれかでなければなりません。*yes* を指定すると、関数 gethostent()、getprotoent()、getnetent()、getservent() のサポートが有効になります。このサポートは、Linux 上で実行中の lfpd 当たり約 500 KB のメモリーを使用します。デフォルトは *yes* です。このパラメーターはオプションです。

VSE_HOSTID
VSE_HOSTID の値は、有効な IPv4 IP アドレスでなければなりません。lfpd はこの IP アドレスをローカル・ホスト ID として使用し、z/VSE アプリケーションが getHostId() 関数を実行する際にはこのアドレスを返します。この設定および RESTRICT_TO_HOSTID により、z/VSE 上で実行するアプリケーションを特定の IP アドレスに「バインド」できます。

VSE_HOSTID6
VSE_HOSTID6 の値は、有効な IPv6 IP アドレスでなければなりません。IPv6 および RESTRICT_TO_HOSTID が有効な場合、このパラメーターは必須です。これらのパラメーターにより、z/VSE 上で実行するアプリケーションが特定の IP アドレスにバインドされます。

DISABLE_IPV6
DISABLE_IPV6 の値は *yes* または *no* でなければなりません。 *yes* を指定すると、lfpd の IPv6 サポートが無効になり、lfpd は IPv4 専用スタックのように動作します。デフォルトは *no* です。このパラメーターはオプションです。

RESTRICT_TO_HOSTID
RESTRICT_TO_HOSTID の値は *yes* または *no* のいずれかでなければなりません。*yes* を指定した場合、`lfpd` は、すべての bind() 呼び出しを、構成済みの VSE_HOSTID のみを使用するように制限します。これにより、z/VSE 上で実行するアプリケーションは、すべての Linux 装置または特定の Linux IP アドレスへのバインドができなくなります。デフォルトは *yes* です。このパラメーターはオプションです。

LOG_INFO_MSG
LOG_INFO_MSG の値は *yes* または *no* のいずれかでなければなりません。デフォルトでは、lfpd はタイプ EMERG、ALERT、および ERR のメッセージをシステム・ロガー (syslogd) に書き込みます。LOG_INFO_MSG を *yes* に設定すると、lfpd は INFO メッセージを追加的にシステム・ロガーに書き込みます。デフォルトは *no* です。このパラメーターはオプションです。

z/VSE

各 LFP インスタンスはその固有の構成を持ちます。構成はインスタンスの開始時に読み取られ、インスタンスの実行中は変更できません。続行する前に、LFP 用のソケット API が、 455 ページの『z/VSE 上でソケット API と Linux Fast Path とを使用するための準備』での説明に従って準備済みであることを確認してください。

LFP オペレーター・ツールの IJBLFPOP は、ユーザーがインスタンスの開始を要求した場合に、構成を処理します。IJBLFPOP は構成の場所をパラメーターとして取ります。この構成は、ライブラリー・メンバー内に備えるか、または IJBLFPOP を呼び出すジョブのジョブ入力 (SYSIPT) に直接備えます。

各構成パラメーターは、`key = value` の形式で指定する必要があります。構成内の空の行は無視されます。行の先頭の文字「*」および「#」はコメント行を示しますが、これも無視されます。構成パラメーターは大/小文字を区別しません。

IUCV 固有の構成
: これらのパラメーターは **IUCV** 接続でのみ使用されます。

IucvMTU|MTU
: IucvMTU (MTU も受け入れ可能) の値は 256 から 65535 の 10 進数でなければなりません。推奨値は 8192 です。IucvMTU サイズは、転送されるパケットの最大サイズを指定します。この値を高くするとパフォーマンスは向上する可能性がありますが、さらに多くの SVA PFIX メモリーが必要になります。IucvMTU 値は、Linux on z Systems 上で実行するターゲット lfpd に指定された IUCV_MTU_SIZE と正確に一致しなければなりません。このパラメーターは必須です。

IucvMsgLimit
: IucvMsgLimit の値は 16 から 65535 の 10 進数でなければなりません。IucvMsgLimit は、Linux 上の `lfpd` によってまだ受け取られていない未解決 IUCV メッセージの最大数を指定します。推奨値は 1024 です。この値を高くするとパフォーマンスが向上する可能性がありますが、さらに多くの SVA PFIX メモリーが必要になります。このパラメーターは必須です。

IucvSrcAppName
: IucvSrcAppName の値は、1 から 8 文字の英数字で指定する必要があります。この値は z/VSE IUCV アプリケーション・テーブル内の項目を定義し、z/VSE システムで固有である必要があります。Linux 上の `lpfd` はこの値を検査して、特定の z/VSE LFP インスタンスだけが Linux 上のこの `lfpd` に接続できることを確認します。このパラメーターは必須です。

IucvDestAppName
: IucvDestAppName の値は、1 から 8 文字の英数字で指定する必要があります。この値は、Linux 上で実行している `lfpd` の IUCV_SRC_APPNAME 構成パラメーターと一致する必要があります。インスタンスはこの IUCV_SRC_APPNAME を持つ `lfpd` と接続します。このパラメーターは必須です。

IucvDestVmId
: IucvDestVmName の値は、1 から 8 文字の英数字で指定する必要があります。この値は、`lfpd` を実行するターゲット Linux on z Systems の z/VM ユーザー ID を指定します。このパラメーターは必須です。

PacketConsolidationThreshold
: PacketConsolidationThreshold の値はバイト単位で指定します。使用可能な最大値は、構成済みの MTU のサイズです。デフォルトは 100 バイトです。このパラメーターを使用して、CPU 消費量とメモリー消費量の最適なトレードオフを設定します。以下の条件が適用されます。

  - ユーザー・データを含む新規パケットが到着した。

- パケット・サイズが PacketConsolidationThreshold を超えていない。
- ターゲット・タスクで別のパケットが既に待ち行列に入れられている。
- 既に待ち行列に入れられているデータ・パケットに、新たに到着したパケットのデータを含めるだけの十分なフリー・スペースがある。
- パケットが属するソケットは STREAM ソケットである。

これらすべての条件が満たされた場合は、新たに到着したパケットのデータ・バイトが、最初のパケットのフリー・スペースにコピーされます。これにより、従来であれば 2 番目のパッケージ用に必要とされていたバッファー・スペースを再利用できるようになりました。このパラメーターはオプションです。

HiperSockets 固有の構成

これらのパラメーターは **HiperSockets** 接続でのみ使用されます。

HsDevices

HsDevices の値は、コンマで区切られた 3 つの装置アドレス (CUU) として指定される必要があります。例: cuu1,cuu2,cuu3 (ここで cuu2 は必ず cuu1+1 とする必要があります)。この装置アドレスは HiperSockets 装置を表している必要があります。また、IPL プロシージャー内では、この装置アドレスを装置タイプ OSAX を使用して定義する必要があります。このパラメーターは必須です。

注**:** HiperSockets 装置の場合、MTU サイズ/フレーム・サイズが IOCP に構成されます。MTU サイズを大きくすると、より多くのバッファー・スペースが必要になる場合があります。IOCP での HiperSockets 装置の構成の詳細については、「*IBM z/VSE* 管理」を参照してください。

HsSrcAppName

HsSrcAppName の値は、1 から 8 文字の英数字で指定する必要があります。この値はローカル・ポートを定義します。この値と HsSrcSystemName の組み合わせが z/VSE システムで固有である必要があります。Linux 上の `lpfd` はこの値を検査して、特定の z/VSE LFP インスタンスだけが Linux 上のこの `lfpd` に接続できることを確認します。このパラメーターは必須です。

HsSrcSystemName

HsSrcSystemName の値は、1 から 8 文字の英数字で指定する必要があります。この値は LFP インスタンスのシステム名を定義します。この名前は、使用される HiperSockets ネットワーク全体で固有である必要があります。内部的に、この値は、HiperSockets 装置の初期化に使用される IPv6 link-local アドレスに変換されます。Linux 上の `lpfd` はこの値を検査して、特定の z/VSE LFP インスタンスだけが Linux 上のこの `lfpd` に接続できることを確認します。このパラメーターは必須です。

HsDestAppName

HsDestAppName の値は、1 から 8 文字の英数字で指定する必要があります。この値は、Linux 上で実行している `lfpd` の HS_SRC_APPNAME 構成パラメーターと一致する必要があります。インスタンスはこの HS_SRC_APPNAME を持つ `lfpd` と接続します。このパラメーターは必須です。

HsDestSystemName

HSDestSystemName の値は、1 から 8 文字の英数字で指定する必要があります。この値は、Linux 上で実行される `lfpd` の HS_SRC_SYSTEMNAME 構成パラメーター (*hsuid*) に一致する必要があります。インスタンスはこの HS_SRC_SYSTEMNAME を持つ `lfpd` と接続します。このパラメーターは必須です。

HsKeepAliveTime

HsKeepAliveTime の値は、秒を表す 10 進数でなければなりません。デフォルトは 5 秒です。z/VSE と Linux 間の接続の予期しない終了の検出にキープアライブ・メカニズムが使用されます。HsKeepAliveTime 値は、キープアライブ・パケットの交換頻度を制御します。このパラメーターはオプションです。

HsMsgLimit

HsMsgLimit の値は 8 から 128 の 10 進数であり、8 の倍数でなければなりません。HsMsgLimit 値は、LFP インスタンスによってまだ受け取られていない未解決メッセージの最大数を指定します。この値を高くするとパフォーマンスは向上する可能性がありますが、さらに多くのメモリーが必要になります。推奨値は 32 です。このパラメーターは必須です。

共通の構成

これらのパラメーターは、**IUCV** 接続と **HiperSockets** 接続の両方で使用できます。

DeviceType

IUCV または *HS* を装置タイプとして指定できます。*IUCV* はデフォルトです。このパラメーターはオプションです。

Id Id パラメーターの値は 2 桁の 10 進数で、00 から 99 の範囲内でなければなりません。この数値は、開始されるインスタンスのインスタンス ID です。各インスタンスは一度だけ開始することができます。したがって、同じ ID のインスタンスが実行することはありません。OPTION SYSPARM='xx' 文が指定されたジョブは、ID 'xx' の LFP インスタンスを使用します。他の TCP/IP スタックも、OPTION SYSPARM 文を用いて、使用するスタックの ID を指定できます。特定の ID を持つあるスタックを開始した場合、その同じ ID の LFP インスタンスを開始することはできません。このパラメーターは必須です。

InitialBufferSpace

InitialBufferSpace の値は、以下のように指定できます。

- バイト数を示す 10 進数として (例えば、512)。
- キロバイト単位で (例えば、1024 K)。
- またはメガバイト単位で (例えば、2 M)。

この値は、スタートアップ時にインスタンスがパケット・バッファーに対して最初に割り振るメモリー・サイズを指定します。メモリーは、ANY 領域で SVA PFIX として割り振られます。最小値は 256 K です。8K の MTU サイズの推奨値は 512 K です。最大値はシステムのレイアウトによって決まります。

注: 必要なバッファー・スペースの量も MTU サイズによって決まります。MTU サイズを大きくすると、より多くのバッファー・スペースが必要になる場合があります。

MaxBufferSpace

MaxBufferSpace の値は、以下のように指定できます。

- キロバイト数を示す 10 進数として (例えば、512)。
- キロバイト単位で (例えば、1024 K)。
- またはメガバイト単位で (例えば、2 M)。

この値は、インスタンスがパケット・バッファーに対して割り振る最大メモリー・サイズを指定します。メモリーは、ANY 領域で SVA PFIX として割り振られます。最小値は、InitialBufferSpace パラメーターの値より低くしてはなりません。推奨値は 4 MB です。最大値は、システムのレイアウトに応じて異なります。

WindowSize

WindowSize の値は 10 進数でなければなりません。この値は、ソケットの受信ウィンドウのサイズを指定します。受信ウィンドウは、アプリケーションが実際にデータをソケットから受け取らなくても受信できるデータの量を定義します。推奨値は 65535 (64 KB) です。この値を高くするとパフォーマンスが向上する可能性がありますが、さらに多くのメモリーが必要になります。最小値は 4096 です。

WindowThreshold

WindowThreshold の値は、パーセンテージを表す 1 から 100 の 10 進数でなければなりません。WindowThreshold は、LFP インスタンスが通知を `lfpd` に送信する前に WindowSize が達する必要があるパーセンテージを指定します。推奨値は 25 です。この値を高くするとパフォーマンスが向上する可能性がありますが、さらに多くのメモリーが必要になります。

サンプル構成

467 ページの図 25 は z/VSE の始動ジョブを示しています。この始動ジョブには、以下のサンプル・エントリーを使用する、IUCV を介した構成が含まれます。

- LFP インスタンスは、Linux システムに z/VM ユーザー ID `LINLFP` で接続する必要があります。
- Linux 上の `lfpd` の IUCV アプリケーション名は `LINR02` です。
- z/VSE 上のインスタンスには、IUCV アプリケーション名 `TESTV` があります。
- z/VSE システムの z/VM ユーザー ID は `VSER05` です。この名前は、Linux 上の `lfpd` 構成で指定されます。

```
* $$ JOB JNM=LFPSTART,CLASS=0,DISP=L
// JOB LFPSTART
// EXEC IJBLFPOP,PARM='START DD:SYSIPT LOGALL'
* Instance ID
ID                 = 01
MTU                = 8192
IucvMsgLimit       = 1024
InitialBufferSpace = 512 K
MaxBufferSpace     = 4M
IucvSrcAppName     = TESTV
IucvDestAppName    = LINR02
IucvDestVMId       = LINLFP
WindowSize         = 65535
WindowThreshold    = 25
/*
/&
* $$ EOJ
```

図 *25. IUCV* を介した *z/VSE LFP* の構成例

Linux 上の lfpd の以下の構成は、上記の z/VSE 上の LFP インスタンス構成と対を成しています。

```
# lfpd configuration file

IUCV_SRC_APPNAME = LINR02

# ensure that only TESTV from VSER05 can connect

PEER_IUCV_VMID     = VSER05
PEER_IUCV_APPNAME  = TESTV

IUCV_MSGLIMIT      = 1024

MTU_SIZE           = 8192
MAX_SOCKETS        = 1024
INITIAL_IO_BUFS    = 128

WINDOW_SIZE        = 65535
WINDOW_THRESHOLD   = 25

VSE_CODEPAGE       = EBCDIC-US

VSE_HOSTID         = 10.0.0.1
RESTRICT_TO_HOSTID = yes

LOG_INFO_MSG       = no
```

図 *26. IUCV* を介した *Linux LFP* の構成例

- IUCV_SRC_APPNAME は、lfpd 構成ファイルの名前を定義します。これは z/VSE 構成ファイル内に定義されている IucvDestAppName と一致している必要があります。
- PEER_IUCV_VMID および PEER_IUCV_APPNAME は、ユーザー ID VSER05 (z/VSE システムの z/VM ユーザー ID) からの TESTV (z/VSE 構成ファイルで定義されている) のみが接続可能になることを保証します。

468 ページの図 27 は z/VSE の始動ジョブを示しています。この始動ジョブには、以下のサンプル・エントリーを使用する、HiperSockets を介した構成が含まれます。

- LFP インスタンスは、Linux システムにシステム名 LINXLPAR で接続する必要があります。

- Linux 内の lfpd のアプリケーション名は LNXSYS1 です。
- z/VSE 上のインスタンスには、アプリケーション名 TESTV があります。
- z/VSE 内の LFP インスタンスのシステム名は VSELPAR です。

```
* $$ JOB JNM=LFPSTART,CLASS=0,DISP=L
// JOB LFPSTART
// EXEC IJBLFPOP,PARM='START DD:SYSIPT LOGALL'
* Instance ID
ID = 02
InitialBufferSpace = 1M
MaxBufferSpace = 4M
WindowSize = 65535
WindowThreshold = 25
DeviceType = HS
HSDevices = 500,501,502
HSMsgLimit = 128
HSSrcAppName = TESTV
HSDestAppName = LNXSYS1
HSSrcSystemName = VSELPAR
HSDestSystemName = LNXLPAR
HSKeepAliveTime = 30
/*
/&
* $$ EOJ
```

図 *27. HiperSockets* を介した *z/VSE LFP* の構成例

Linux 上の lfpd の以下の構成は、上記の z/VSE 上の LFP インスタンス構成と対を成しています。

```
# lfpd sample configuration file
#

DEVICETYPE         = HS

HS_MSGLIMIT        = 128

HS_SRC_APPNAME     = LNXSYS1
HS_SRC_SYSTEMNAME  = LNXLPAR

# ensure that only TESTV from VSELPAR can connect

PEER_HS_APPNAME    = TESTV
PEER_HS_SYSTEMNAME = VSELPAR

MAX_SOCKETS        = 1024
INITIAL_IO_BUFS    = 128

WINDOW_SIZE        = 65535
WINDOW_THRESHOLD   = 25

VSE_CODEPAGE       = EBCDIC-US

VSE_HOSTID         = 10.0.0.1
RESTRICT_TO_HOSTID = no

LOG_INFO_MSG = no
```

図 *28. HiperSockets* を介した *Linux LFP* の構成例

- HS_SRC_APPNAME は、lfpd 構成ファイルの名前を定義します。これは z/VSE 構成ファイル内に定義されている HSDestAppName と一致している必要があります。

- PEER_HS_SYSTEMNAME と PEER_HS_APPNAME を使用することで、ユーザー VSELPAR (z/VM ゲスト) からの TESTV (z/VSE 構成ファイルに定義済み) のみが接続できるようになります。

Linux ファスト・パスの開始と停止

z/VSE

LFP インスタンスの開始

z/VSE 上で LFP インスタンスを開始するには、LFP オペレーター・プログラム IJBLFPOP を使用します。起動パラメーター・フォーマットは次のとおりです。

```
EXEC IJBLFPOP,PARM='START <CFGFILE> [LOGALL]'
```

- すべてのパラメーターは定位置であり、表示順序で表示される必要があります。
- CFGFILE は必須パラメーターであり、IJBLFPOP のインスタンス構成の読み取り元を指定します。
- CFGFILE は、DD: が先行する、有効な Language Environment ファイル名で宣言される必要があります。
- 構成がライブラリー PRD2.CONFIG 内のメンバー LFPCFG00.L である場合は、CFGFILE を DD:PRD2.CONFIG(LFPCFG00.L) と指定します。
- 構成が IJBLFPOP を実行するジョブ内にある場合は、CFGFILE を DD:SYSIPT と指定します。
- LOGALL はオプション・パラメーターです。このパラメーターが指定されると、IJBLFPOP はすべてのメッセージをコンソールに書き込みます。このパラメーターが省略されると、エラー・メッセージのみがコンソールに出力されます。すべてのメッセージは必ずジョブ・リスト内に出力されます。
- ICCF ライブラリー 59 にあるスケルトン SKLFPSTA は、ジョブそのものの内部にインスタンス構成が入る、LFP インスタンス・スタートアップ・ジョブを作成するために使用できます。

以下のメッセージがコンソールに表示される場合は、インスタンスは正常に開始されています。

```
LFPB013I  STARTED LFP INSTANCE '00'
```

LFP インスタンスの停止

z/VSE 上で LFP インスタンスを停止するには、LFP オペレーター・プログラム IJBLFPOP を使用します。起動パラメーター・フォーマットは次のとおりです。

```
EXEC IJBLFPOP,PARM='STOP <INSTID>'
```

- すべてのパラメーターは定位置であり、表示順序で表示される必要があります。
- INSTID は、停止するインスタンスの ID です。
- インスタンスが実行中の場合、IJBLFPOP は、インスタンスの停止を実行するために確認する必要があるメッセージをコンソールに送信します。インスタンスを停止するには YES を、取り消すには NO を、現在そのインスタンスを使用しているすべてのタスクおよびそのパーティションをリストするには LIST を入力できます。
- インスタンスはいつでも停止できます。

- インスタンスを使用しているアクティブ・タスクは、停止したインスタンスの関数を呼び出す際に、適切なエラー・コードを取得します。
- ICCF ライブラリー 59 のスケルトン SKLFPSTO は、特定の LFP インスタンスを停止するジョブを作成するために使用できます。

以下に示すのは、正常に停止したインスタンスの例です。

```
LFPB012I  REALLY STOP INSTANCE '01' (3 TASKS ACTIVE)? (YES/NO/LIST)
0 list
LFPB028I  ACTIVE TASK IDS FOR INSTANCE '01':
       2E (Z1)     2F (Z2)     30 (Z3)
LFPB029I  END OF LIST OF ACTIVE TASK IDS.
LFPB012I  REALLY STOP INSTANCE '01' (3 TASKS ACTIVE)?(YES/NO/LIST)
0 yes
LFPB020I  STOPPED LFP INSTANCE '01'.
```

Linux on z Systems

LFP デーモンは、lfpd を直接呼び出すことにより、または制御スクリプト lfpd-ctl を使用することにより、開始および停止することができます。通常は、制御スクリプトが lfpd を制御する適切な方法を示しているので、それのみが使用されます。ただし、デバッグなどのタスクの場合、制御スクリプトを使用せずに lfpd を直接開始する必要があります。続くいくつかのセクションでは、以下の方法について説明しています。

- 制御スクリプトを使用して、lfpd を制御します。
- 制御スクリプトを使用せずに、lfpd を開始します。
- 制御スクリプトを使用せずに、lfpd を停止します。

制御スクリプトを使用した LFP デーモンの制御

lfpd 制御スクリプト lfpd-ctl は、すべての有効な構成ファイルが、 457 ページの『Linux Fast Path の構成』で説明されている命名規則を順守することを要求します。lfpd-ctl は、/etc/opt/ibm/vselfpd/confs-enabled ディレクトリー内でリンクされている構成のみを開始できます。

lfpd-ctl の呼び出しパラメーターは、以下のとおりです。

```
lfpd-ctl (start|stop|restart|force-reload|status|startdbg) <APPNAME>
lfpd-ctl (list|startall|stopall)
```

lfpd-ctl start APPNAME
このコマンドは、指定された構成で LFP デーモンを開始します。上記のコマンドは、例えば、lfpd を構成ファイル /etc/opt/ibm/vselfpd/confs-enabled/lfpd-APPNAME.conf を使用して呼び出します。lfpd のスタートアップに失敗した場合は、Linux システム・ログ (通常はファイル /var/log/messages) で、スタートアップの失敗に関連するメッセージがないかどうかを確認してください。その代わりに、後述のように、コマンド lfpd-ctl startdbg APPNAME を使用することもできます。

lfpd-ctl stop APPNAME
このコマンドは、APPNAME という名前の構成で開始された、実行中の LFP デーモンを停止します。

lfpd-ctl restart | force-reload APPNAME
restart および force-reload パラメーターは、lfpd を停止し、次いでそ

れを再開します。これらは構成が変更されている場合にのみ使用してください。実行中の lfpd に動的に適用することはできません。

lfpd-ctl status APPNAME

status コマンドは、lfpd の状況を示します。状況は以下のいずれかになります。

- 実行していない
- z/VSE LFP インスタンスの接続を *listen* 中
- *xxx* に接続済み。ここで *xxx* は、接続された z/VSE LFP インスタンスのアプリケーション名です。

lfpd-ctl startdbg APPNAME

このコマンドは、指定された構成で lfpd を開始します。この機能は lfpd-ctl start APPNAME と似ています。ただし、lfpd のスタートアップが失敗すると、コマンドの発行元のコンソールに、関連する lfpd のメッセージが表示されます。Linux システム・ログでメッセージを確認する必要はありませんが、システム・ログにもメッセージが示されます。

lfpd-ctl list

list コマンドは、実行中または使用可能な各 lfpd と、デーモンのそれぞれの状況に関する情報が含まれるテーブルを生成します。

lfpd-ctl startall

startall コマンドは、まだ開始していないすべての使用可能な構成を開始しようとします。このコマンドは、システムのスタートアップ時にも、SysV 開始スクリプトによって呼び出されます。

lfpd-ctl stopall

stopall コマンドは、それが引き続き使用可能であるかどうかに関係なく、実行中のすべての lfpd を停止します。このコマンドは、システムのシャットダウン時や、lfpd の rpm インストール・パッケージがシステムから削除される場合にも、SysV 開始スクリプトによって呼び出されます。

制御スクリプトを使用しない LFP デーモンの開始

LFP デーモンを Linux on z Systems 上で開始するには、lfpd プログラムを実行します。Lfpd は、バックグラウンドで実行でき、z/VSE システム上の 1 つの LFP インスタンス用に機能するデーモンです。起動パラメーター・フォーマットは次のとおりです。

```
lfpd <--file|-f configfile> [--debug|-d] [--packets|-p] [--startuplogfile|-s logfile]
[--help|-h]
```

- パラメーターは定位置ではなく、任意の順序で指定できます。
- configfile は必須パラメーターであり、lfpd の構成ファイルの絶対パスを指定します。例えば、/etc/lfpd/lfpcfg.cfg となります。
- --debug または -d を指定した場合、lfpd はバックグラウンドでは実行されませんが、デバッグ・メッセージを呼び出し元のコンソールに書き込みます。デバッグ出力は、サポート担当者が問題を追跡するためにのみ必要です。
- -packets または -p を指定した場合、lfpd はバックグラウンドでは実行されませんが、パケット・ダンプを呼び出し元のコンソールに書き込みます。z/VSE シ

ステムとの間で転送されるすべてのパケットは、テキスト形式で表示されます。通常、パケット・ダンプは、サポート担当者が問題を追跡するためにのみ必要です。

- lfpd を呼び出した後に、Linux システム・ログ (通常は /var/log/messages) で、lfpd からのエラー・メッセージまたはその始動メッセージを調べ、lfpd が正しく開始していることを確認します。
- --startuplogfile または -s logfile が指定された場合、lfpd は、指定のログ・ファイルにすべてのスタートアップ・メッセージを追加で記録します。

```
linlfp lfpd[6185]: Beginning startup of lfpd
linlfp lfpd[6185]: Now listening on application name 'LINR02'
linlfp lfpd[6185]: Accepted new connection on LINR02   from VSER05
/TESTV   , new socketfd=11
linlfp lfpd[6185]: Configuration matched, accepting connection.
```

図 *29. Linux* システム・ログ出力の例

制御スクリプトを使用しない LFP デーモンの停止

lfpd は通常はバックグラウンドで実行します。lfpd を終了するには、SIGTERM シグナルを、終了する必要がある実行中の lfpd の pid (プロセス ID) に送信します。ローカル・アプリケーション名 (lfpd 構成内の HS_SRC_APPNAME または IUCV_SRC_APPNAME) は、実行中の lfpd を一意的に識別します。停止させる lfpd の pid を判別するには、以下の 2 つの方法があります。

1. システム・ログ・メッセージ・ファイル (通常は /var/log/messages) を開き、以下のような行を調べます。

   ```
   linlfp lfpd[6185]: Now listening on application name 'LINR02'
   ```

   値 6185 は、アプリケーション名 LINR02 を使用する lfpd の pid です。
2. 別の方法として、実行中の lfpd のロック・ファイルを使用して、その pid を検出できます。実行中の各 lfpd は、ロック・ファイルを /var/run ディレクトリー内に作成します。例えば、アプリケーション名 LINR02 のロック・ファイル名は、lfpd-LINR02.pid です。ロック・ファイルの内容は、lfpd プロセスの pid です。

SIGTERM シグナルを実行中の lfpd に送信する前に、現在 lfpd に接続されている z/VSE システムがないことを確認してください。lfpd を終了しても、z/VSE システムが引き続き接続されている場合は、z/VSE 上の LFP インスタンスがそれを検出して、その固有の状況を DOWN に設定します。

/var/log/messages に進み、z/VSE システムが実行中の lfpd から切断されていることを確認します。

```
linlfp lfpd[6185]: All tasks abended, status set to DOWN.
linlfp lfpd[6185]: Data socket closed, ready for new connection on
application name 'LINR02'.
```

図 *30.* 切断された *z/VSE* システムの *Linux* システム・ログ出力例

これが pid 6185 の lfpd からの最終出力である場合、現在接続されている z/VSE システムはなく、lfpd の終了は、z/VSE 上で実行しているアプリケーションによる予測不能な結果とはなりません。

管理用タスク

z/VSE

アクティブ・インスタンスのリスト

z/VSE システム上の現在のアクティブ・インスタンスのリストを表示するには、LFP オペレーター・プログラム IJBLFPOP を使用してください。

起動パラメーター・フォーマットは次のとおりです。

```
EXEC IJBLFPOP,PARM='LIST'
```

出力は必ず、コンソールとジョブ・リストに書き込まれます。

ICCF ライブラリー 59 のスケルトン SKLFPLST は、実行中のすべての LFP インスタンスをリストするジョブを作成するために使用できます。

```
LFPB025I  ACTIVE LFP INSTANCES: 1
    INSTANCE 01 HAS   3 ACTIVE TASKS
LFPB026I  END OF ACTIVE LFP INSTANCES LIST.
```

図 *31.* アクティブ・インスタンスのリスト出力例

アクティブ・インスタンスに関する情報の表示

z/VSE 上の LFP インスタンスに関する詳細情報を表示するには、LFP オペレーター・プログラム IJBLFPOP を使用します。

起動パラメーター・フォーマットは次のとおりです。

```
EXEC IJBLFPOP,PARM='INFO <INSTID> [SHOWTASKS] [LOGALL]'
```

- すべてのパラメーターは定位置であり、表示順序で表示される必要があります。
- INSTID は、その情報を表示するインスタンスの ID です。
- SHOWTASKS はオプション・パラメーターです。このパラメーターを使用して、現在 LFP を使用している各タスクに関する詳細情報を表示します。
- LOGALL はオプション・パラメーターです。このパラメーターが指定されると、IJBLFPOP はすべてのメッセージをコンソールに書き込みます。このパラメーターが省略されると、エラー・メッセージのみがコンソールに出力されます。すべてのメッセージは必ずジョブ・リスト内に出力されます。
- 表示された情報は、実行中のインスタンスのさまざまな値を示します。その一部は、現在のメモリー消費量と、アクティブ・タスクやそのソケットに関する詳細を示します。他の値は、サポート担当者が問題を追跡するためにのみ検討の対象になります。
- ICCF ライブラリー 59 のスケルトン SKLFPINF は、特定の LFP インスタンスに関する情報を表示するジョブを作成するために使用できます。

以下に示すのは、すべてのパラメーターが設定された IUCV 接続を介した LFP の出力例です。

図 *32. IUCV* 接続を介した *z/VSE* 上の *LFP* インスタンスの出力例

```
LFPB023I  INFO ABOUT LFP INSTANCE '01':
  *** INSTANCE ***
    STATUS ....................... : UP
    WINDOW SIZE .................. : 65,535
    WINDOW THRESHOLD ............. : 25%  (16,383 bytes)
    TASKS WAITING FOR MSGLIMIT ..... : 0
    TIMES IN WAIT FOR MSGLIMIT ..... : 0
    TASKS WAITING FOR PACKET ....... : 0
    TIMES IN WAIT FOR PACKET ....... : 0
    GETVIS POOL ID ................. : ILFP01
  *** DEVICE ***
    DEVICE STATUS .................. : ACTIVE
    DEVICE TYPE .................... : IUCV
    MTU SIZE ....................... : 8,192
    PACKETS SENT ................... : 88
    PACKETS RECEIVED ............... : 53
    PACKETS DROPPED ................ : 0
    PACKETS WAITING FOR MSG COMPLETE : 0
    FREE NOTIFY BUFFERS ............ : 1,024
    CONFIGURED LOCAL IUCV MSGLIMIT . : 1,024
    ACTUAL LOCAL IUCV MSGLIMIT ..... : 1,024
    ACTUAL REMOTE IUCV MSGLIMIT .... : 1,024
    IUCV MSGLIMIT EXCEEDED ......... : 0
    IUCV SOURCE APPL. NAME ......... : TESTV
    IUCV DEST. APPL. NAME .......... : LFP61
    IUCV DEST. VM USERID ........... : LINLFP
    IUCV DEST. VM SYSTEM ........... : (LOCAL)
  *** BUFFER MANAGER ***
    CURRENTLY USED MEMORY .......... : 524,160
    INITIAL MEMORY SIZE ............ : 524,288
    MAXIMUM MEMORY SIZE ............ : 4,194,304
   -- POOL #1 --
    PACKET SIZE .................... : 64
    INITIAL POOL SIZE .............. : 181,568
    CURRENT POOL SIZE .............. : 181,440
    POOL COUNT ..................... : 1
    AVAILABLE PACKET COUNT ......... : 266
    OVERALL PACKET COUNT ........... : 266
    MAXIMUM PACKETS USED ........... : 2
   -- POOL #2 --
    PACKET SIZE .................... : 3,552
    INITIAL POOL SIZE .............. : 178,864
    CURRENT POOL SIZE .............. : 176,160
    POOL COUNT ..................... : 1
    AVAILABLE PACKET COUNT ......... : 43
    OVERALL PACKET COUNT ........... : 43
    MAXIMUM PACKETS USED ........... : 2
   -- POOL #3 --
    PACKET SIZE .................... : 8,192
    INITIAL POOL SIZE .............. : 174,762
    CURRENT POOL SIZE .............. : 166,560
    POOL COUNT ..................... : 1
    AVAILABLE PACKET COUNT ......... : 18
    OVERALL PACKET COUNT ........... : 19
    MAXIMUM PACKETS USED ........... : 6
  *** DEBUG ***
    DEBUG IS ENABLED
    DEBUG AREA SIZE ................ : 320,000
    DEBUG ENTRY COUNT .............. : 10,000
  *** DIAGNOSIS ***
    DIAGNOSIS IS DISABLED
  *** TASKS ***
    ACTIVE TASK COUNT .............. : 1
   -- TASK #1 --
    TASK ID (PARTITION ID) ......... :   2D (R1)
    SOCKET COUNT ................... : 2
    L2 SOCKET LIST COUNT ........... : 1
```

```
    PENDING REQUESTS ............... : 0
    PENDING SOCKET REQUESTS ........ : 0
    REQUESTS WAITING FOR MSGLIMIT .. : 0
    TIMES IN WAIT FOR MSGLIMIT ..... : 0
    PACKETS WAITING FOR RECEIVE .... : 0
    TIMES IN WAIT FOR PACKET ....... : 0
    FREE REQUEST COUNT ............. : 2
LFPB024I  END OF INFO ABOUT LFP INSTANCE '01'.
```

以下に示すのは、すべてのパラメーターが設定された HiperSockets 接続を介した LFP の出力例です。

図 *33. HiperSockets* 接続を介した *z/VSE* 上の *LFP* インスタンスの出力例

```
LFPB023I  INFO ABOUT LFP INSTANCE '99':
  *** INSTANCE ***
    STATUS ......................... : UP
    WINDOW SIZE .................... : 65,535
    WINDOW THRESHOLD ............... : 25%  (16,383 bytes)
    PACKET CONSOLIDATION THRESHOLD . : 100
    TASKS WAITING FOR CQ OR WINDOW . : 0
    TIMES IN WAIT FOR CQ OR WINDOW . : 0
    TASKS WAITING FOR PACKET ....... : 0
    TIMES IN WAIT FOR PACKET ....... : 0
    GETVIS POOL ID ................. : ILFP99
  *** DEVICE ***
    DEVICE STATUS .................. : ACTIVE
    DEVICE TYPE .................... : HS
    MTU SIZE ....................... : 8,192
    PACKETS SENT ................... : 4
    PACKETS RECEIVED ............... : 3
    PACKETS DROPPED ................ : 0
    HS COMPLETION QUEUE STATES ..... : 0
    PROTOCOL WINDOW FULL STATES .... : 0
    QUEUE LENGTH ................... : 16
    PROTOCOL WINDOW LENGTH ......... : 10
    KEEP-ALIVE TIME SECONDS ........ : 999,999
    HS DEVICES ..................... : E00,E01,E02
    HS SOURCE APPL. NAME ........... : TESTV2
    HS SOURCE SYSTEM NAME .......... : VSELFP61
    HS DEST. APPL. NAME ............ : LFP612
    HS DEST. SYSTEM NAME ........... : LINHSI0
  *** BUFFER MANAGER ***
    .
    .     ...same output as for IUCV ...
    .
 LFPB024I  END OF INFO ABOUT LFP INSTANCE '99'
```

ソケット診断の有効化または無効化

z/VSE 上の LFP インスタンスに関する詳細情報を表示するには、LFP オペレーター・プログラム IJBLFPOP を使用します。

起動パラメーター・フォーマットは次のとおりです。

```
EXEC IJBLFPOP,PARM='DIAG <INSTID> <ON|OFF>'
```

- すべてのパラメーターは定位置であり、表示順序で表示される必要があります。
- INSTID は、ソケット診断設定を変更するインスタンスの ID です。
- ON を指定するとソケット診断は有効になり、OFF を指定すると無効になります。LFP インスタンスに対してソケット診断が ON である場合、LE/C ソケッ

ト API ($EDCTCPV) を使用するアプリケーションによってソケットが閉じられるたびに、ソケット統計がジョブ・リストに出力されます。

以下に示すのは、ソケット診断の出力例です。

図 *34.* ソケット診断の出力例

```
LFP Statistics for Socket '0':
  Number of Bytes Sent:             1020364776
  Number of Bytes Received:         943718400
  Number of Packets Sent:           233923
  Number of Packets Received:       183220
  Times in Wait for Send Window:    18158
  Times in Wait for Receive Window: 28002
  Times in Wait for Message Limit:  0
  Times in Wait for Packets:        0
```

Linux on z Systems

LFP デーモン状況の表示

実行中の LFP デーモンの状況を表示するには、lfpd-admin プログラムを使用します。

起動パラメーター・フォーマットは次のとおりです。

```
lfpd-admin <--appname|-appname> <--status|-s>
```

- パラメーターは定位置ではなく、任意の順序で指定できます。
- appname には、実行中の lfpd が使用する HS_SRC_APPNAME または IUCV_SRC_APPNAME を入力します。HS_SRC_APPNAME および IUCV_SRC_APPNAME は lfpd 構成のパラメーターです。
- status コマンドは、lfpd-admin が呼び出されたコンソール上でさまざまな状況情報を表示します。
- lfpd-admin プログラムを使用して、ファイルへのデバッグ・トレースを開始することもできます。この機能は通常は、必要時にパラメーターに関する指示を送信する、サポート担当者のみが使用します。

以下に示すのは、IUCV を介した lfpd-admin の出力例です。

図 *35. IUCV* 接続を介した *lfpd-admin* の出力例

```
linlfp:~ # lfpd-admin -a lfp61 -s
Configuration:
--------------
  DEVICETYPE ....... = IUCV
  LOCAL_IUCV_APPNAME = LFP61
  PEER_IUCV_VMID ... = VSELFP61
  IUCV_MSGLIMIT .... = 1024
  IUCV_MTU_SIZE .... = 8192
  MAX_VSE_TASKS .... = 512
  MAX_SOCKETS ...... = 1024
  INITIAL_IO_BUFS .. = 128
  WINDOW_SIZE ...... = 65535
  WINDOW_THRESHOLD . = 25%  (16383 bytes)
  LINUX_CODEPAGE ... = '850'
  VSE_CODEPAGE ..... = 'EBCDIC-US'
  SUPPORT_GETXXXENT  = yes
```

```
  VSE_HOSTID ....... = 9.152.84.210
  RESTRICT_TO_HOSTID = no
  LOG_INFO_MSG ..... = no

Status:
-------
  z/VSE instance is connected.
  Peer system name ... : VSELFP61
  Peer appl. name .... : TESTV
  Actual IUCV MSGLIMIT : 1024

  Applied host id .... : 9.152.84.210
  Applied host name .. : linlfp

  Allocated I/O buffers ....... : 128
  Free I/O buffers ............ : 128
  Allocated request buffers ... : 128
  Free request buffers ........ : 126
  Allocated socket buffers .... : 128
  Free socket buffers ......... : 125
  Buffers waiting for send .... : 0

  Number of active z/VSE tasks  : 1
  Number of active sockets .... : 2

Trace Status:
-------------
  Running in daemon mode
  No trace is running
```

以下に示すのは、HS CQ を介した lfpd-admin の出力例です。

図 *36. HS CQ* 接続を介した *lfpd-admin* の出力例

```
[root@linhsi0 ~]# lfpd-admin -a lfp612 -s
Configuration:
--------------
  DEVICETYPE ....... = HS
  HS_SRC_APPNAME ... = LFP612
  HS_SRC_SYSTEMNAME  = LINHSI0
  HS_MSGLIMIT ...... = 128
  MAX_VSE_TASKS .... = 512
  MAX_SOCKETS ...... = 1024
  INITIAL_IO_BUFS .. = 128
  WINDOW_SIZE ...... = 65535
  WINDOW_THRESHOLD . = 25%  (16383 bytes)
  LINUX_CODEPAGE ... = '850'
  VSE_CODEPAGE ..... = 'EBCDIC-US'
  SUPPORT_GETXXXENT  = yes
  VSE_HOSTID ....... = 9.152.131.42
  RESTRICT_TO_HOSTID = no
  LOG_INFO_MSG ..... = no

Status:
-------
  z/VSE instance is connected.
  Peer system name ... : VSELFP61
  Peer appl. name .... : TESTV2

  Applied host id .... : 9.152.131.42
  Applied host name .. : linhsi0

  MTU size ..................... : 8088
  Allocated I/O buffers ....... : 128
  Free I/O buffers ............ : 128
```

```
  Allocated request buffers ... : 128
  Free request buffers ........ : 128
  Allocated socket buffers .... : 128
  Free socket buffers ......... : 127
  Buffers waiting for send .... : 0

  Number of active z/VSE tasks  : 0
  Number of active sockets .... : 0

Trace Status:
-------------
  Running in daemon mode
  No trace is running
```

第 15 章 z/VSE - z/VM IP アシスト

概要

z/VSE - z/VM IP アシスト機能 (単に z/VSE VIA とも呼ばれる) は、z/VSE の Linux ファスト・パスに対応する z/VM を提供します。「従来」の z/VSE Linux Fast Path機能では、Linux Fast Path デーモン (`lfpd`) を実行するのに、ユーザーは Linux on z Systems システムのインストール、管理、および構成を行う必要がありました。この Linux on z Systems システムでは、`lfpd` をインストールして構成する必要があります。Linux on z Systems システムのインストールと保守をお客様自身で行うことを望まない場合は、z/VSE VIA 機能が用意されています。これは、使いやすく、すぐに実行できる z/VM ゲスト・イメージを提供するものであり、z/VSE Linux Fast Path の使用に必要なすべてのサービスが提供されます。

最少要件

- IBM zEnterprise® システム (z196 または z114)、ドライバー・レベル 86 以降
- z/VM 6.1 以降
- z/VSE 5.1

z/VSE VIA を使用する場合の通信フロー

図 37 は、IUCV 接続上で z/VSE VIA を使用する通信フローを示しています。

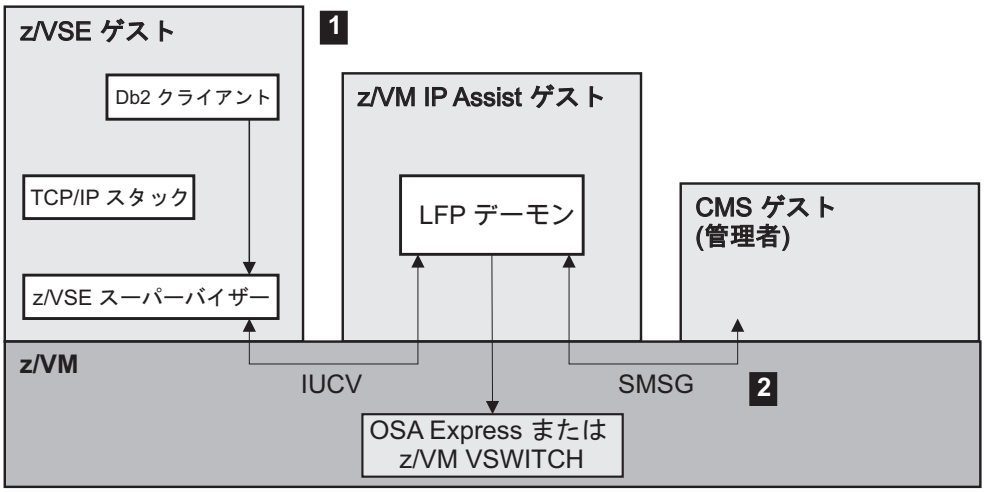


図 *37. z/VSE VIA* および *IUCV* 接続を使用する通信

図 37 で示す処理は、以下のとおりです。

1 IUCV を介した通信は 450 ページの図 23 の説明と同じです。

2 管理者は、SMSG メッセージを CMS ゲストから z/VSE VIA ゲストに送信することで、z/VSE VIA ゲストと対話します。

注: SMSG コマンドおよび VSWITCH コントローラーの詳細については、z/VM の資料を参照してください。

z/VSE VIA z/VM ゲスト構成

z/VSE VIA ゲスト・イメージは、z/VM ディレクトリー・エントリーの LOADDEV ディレクトリー制御ステートメントの SCPDATA パラメーターを使用して構成する必要があります。SCPDATA パラメーターには、z/VSE VIA ゲスト用に使用されるネットワーク構成を指定します。情報は JSON (JavaScript Object Notation。詳しくは、www.json.org を参照) 形式で設定されます。SCPDATA は、EBCDIC コード・ページ **924** を使用して指定する必要があります。通常、z/VSE VIA ゲストは、z/VM の開始時に自動的に始動するように構成されます。

z/VSE VIA ゲストは、次の 2 つの CMS ミニディスクにアクセスできるように構成されています。

構成ディスク
: このミニディスクには、`lpfd` 構成ファイル (`lpfd` インスタンスごとに 1 つ) と SENDERS.ALLOWED ファイルが含まれます。これは、読み取り専用モードで `0D4C` としてリンクされる必要があります。構成ディスクは、始動時に z/VM ディレクトリー・エントリーによってリンクされる必要があります。通常、構成ディスク上のファイルを使用する構成は、管理者の CMS ゲストによって実行されます。したがって、この CMS ゲストは、読み取り/書き込みモードで構成ディスクにアクセスできる必要があります。

データ・ディスク
: ユーザーがトレースを開始すると、トレース・ファイルがこのミニディスクに書き込まれます。ミニディスクはオプションです。これが必要になるのは、ユーザーがトレースを開始する必要がある場合のみです。これは、読み取り/書き込みモードで `0D4D` としてリンクされる必要があります。データ・ディスクには始動時にリンクできます。または、トレースを開始する必要がある場合に次の CP コマンドを使用してリンクできます。

```
CP LINK ZVSEVIA 4321 0D4D MR
```

SCPDATA パラメーターでは、以下のエレメントがサポートされます。

profiles
: プロファイル名を指定します。「zVSE-VIA」を指定する必要があります。このエントリーは必須です。

hostName
: 使用するホスト名が含まれるストリングとして指定する必要があります。ホスト名が指定されていない場合、1 番目のプロファイルの名前が使用されます。この項目はオプションです。

networkCards
: ゲストで使用する、0 から n のネットワーク・カードの配列を定義します。各配列エレメントは、次の 2 つのエレメントを含む JSON オブジェクトです。

- カード・タイプ: `OSA`、`OSM`、`OSX`、`HiperSockets` のいずれかの後に、各装置のアドレスまたは `all` を続ける。`all` を指定した場合、同じタイプの他の装置は定義できません。

- **IP** 構成: linkLocalIPv6、staticIPv4、staticIPv6 のいずれかの後に、IP アドレスまたは null (他の構成の必要がない場合) を続ける。

同じカードに対して IPv4 構成や IPv6 構成を定義できます。この項目はオプションです。

defaultGateway
使用するデフォルト・ゲートウェイが含まれるストリング。この項目はオプションです。

DNS
DNS サーバーの IP アドレスが含まれるストリングの配列。この項目はオプションです。

注**:** LOADDEV SCPDATA 制御ステートメントの詳細については、z/VM の資料を参照してください。z/VM での命名規則は、z/VSE の場合とは異なり、「SCPDATA パラメーター」ではなく、「SCPDATA オペランド」であることに注意してください。

例

次に、z/VSE VIA ゲスト用のユーザー・ディレクトリー・エントリーの例を示します。

```
USER ZVSEVIA  AUTOONLY 1G 1G G
  COMMAND SET D8ONECMD * OFF
  COMMAND SET RUN ON
  COMMAND TERM LINEND #
  COMMAND SPOOL CONS START *
  IPL _0___1__
  LOADDEV PORT 0
  LOADDEV LUN 0
  LOADDEV BOOT 0
  LOADDEV BR_LBA 601
* Network adapters and configuration
  LOADDEV SCPDATA '{',
    '"profiles":["zVSE-VIA"],',
    '"networkCards": [',
  { "OSM": "all", "linkLocalIPv6": null},',
  { "OSA": "2408", "staticIPv4": "9.152.11.86/24"},',
  { "OSX": "110", "staticIPv6": "2001:0db8:85a3::7334/64"},',
  { "hipersockets": "9000", "linkLocalIPv6": null},',
  ],',
    '"defaultGateway":"y.y.y.y/nn",',
    '"DNS":["y.y.y.y/nn","z.z.z.z/nn"],',
    '"hostName":"myServer"',
    '}'
* machine type and number of CPUs
  MACH XA 1
  OPTION LXAPP LANG AMENG
  OPTION MAXCONN 128
* IUCV authorizations
  IUCV ANY PRIORITY MSGLIMIT 1024
  IUCV ALLOW
* Standard virtual devices
  CONSOLE 009 3215 T
  SPOOL 000C 2540 READER *
  SPOOL 000D 2540 PUNCH A
  SPOOL 000E 1403 A
* Network definitions (use either DEDICATE or NIC+VSWITCH)
  DEDICATE dddd vvvv
* COMMAND DEFINE NIC vvvv TYPE QDIO
* COMMAND COUPLE vvvv TO SYSTEM vswitch
```

```
* Minidisks
  LINK MAINT 0190 0190 RR
  LINK MAINT 019D 019D RR
  LINK MAINT 019E 019E RR
* 191 minidisk is optional
* MDISK 191 3390 XXXX 007 AUTOV  MR XXXXXXXX XXXXXXXX XXXXXXXX
* Disk for configuration (D4C) and log file (D4D)
  MDISK D4C 3390 XXXX 009 AUTOV  RR XXXXXXXX XXXXXXXX XXXXXXXX
  MDISK D4D 3390 XXXX 071 AUTOV  MR XXXXXXXX XXXXXXXX XXXXXXXX
```

z/VSE VIA Linux ファスト・パスの構成

z/VSE VIA イメージを実行する z/VM ゲストの他に、z/VSE VIA ゲストが実行する必要がある 1 つ以上の lfpd も構成する必要があります。lfpd は、構成ミニディスク (0D4C) に存在するファイルに加えて、z/VSE VIA が実行されている z/VM ゲストにユーザーが送信する SMSG を使用して構成されます。

管理者は、SMSG メッセージを CMS ゲストから z/VSE VIA ゲストに送信することで、z/VSE VIA ゲストと対話できます。無許可の SMSG トラフィックを防止するために、許可ユーザーのリストに対して各送信者の妥当性検査が行われます。このリストは、構成ディスク (0D4C) 上にある CMS ファイル「SENDERS.ALLOWED」に含まれます。このファイルの各行に 1 つの z/VM ユーザー ID が含まれます。指定されたすべての ID に、z/VSE VIA ゲストへの SMSG コマンドの送信権限があります。

lfpd の構成ファイルは、構成ディスク (0D4C) にも存在する必要があります。構成ファイルの内容は、 470 ページの『Linux on z Systems』で説明されているものと同じです。

各構成ファイルのファイル名は、構成する IUCV_SRC_APP と同じファイル名にする必要があり、各構成ファイルのファイル・タイプは CMS ファイル・タイプ「LFPDCONF」にする必要があります。z/VSE VIA ゲストの始動時に構成ディスクに存在するすべての構成が自動で開始されます。z/VSE VIA ゲストの始動が完了すると、ユーザーは、サポートされる SMSG コマンドを使用して構成を管理できます。

構成ディスクの内容の例:

```
CONFIG-1  LFPDCONF
CONFIG-2  LFPDCONF
SENDERS   ALLOWED
```

z/VSE VIA Linux ファスト・パスの管理

通常、lfpd の管理は、LFP 制御スクリプト lfpd-ctl および lfpd-admin プログラムを使用して行います。z/VSE VIA 機能の場合、SMSG を使用して lfpd を管理します。1 つの z/VM ゲストと別の z/VM ゲスト間でテキスト・メッセージを送信するには、SMSG を使用します。z/VSE VIA 機能にはユーザー検査システムが実装されています。これにより、権限のあるユーザーのみが lfpd 管理インターフェースにアクセスできます。

SMSG を z/VSE VIA 管理インターフェースに送信するための一般的な構文は次のとおりです。

```
SMSG <ZVSEVIA> APP <CMD PARAMS>
```

管理コマンドは、その出力を SMSG コマンドの発行者に 1 行ずつ送信することで出力を返します。

図 *38. z/VSE VIA*管理コマンドの出力例

```
smsg vsevia app lfpd
Ready; T=0.01/0.01 15:39:24
 15:34:24  * MSG FROM VSEVIA  : USAGE: SMSG <GUEST> APP LFPD (START|STOP|RESTART|FORCE-RE
LOAD|STATUS|STARTDBG) <IUCVNAME>
 15:34:24  * MSG FROM VSEVIA  :        SMSG <GUEST> APP LFPD (LIST|STARTALL|STOPALL)

smsg vsevia app lfpd-admin
Ready; T=0.01/0.01 15:39:59
 15:34:59  * MSG FROM VSEVIA  : USAGE: SMSG <GUEST> APP LFPD-ADMIN <IUCVNAME> TRACE START
<FILENAME> (DEBUG|PACKETS|ALL) (SINGLE|WRAP) (MAXSIZE)
 15:34:59  * MSG FROM VSEVIA  :        SMSG <GUEST> APP LFPD-ADMIN <IUCVNAME> TRACE STOP
 15:34:59  * MSG FROM VSEVIA  :        SMSG <GUEST> APP LFPD-ADMIN <IUCVNAME> STATUS
```

管理コマンドは次のとおりです。

- **LFPD**: lfpd-ctl スクリプトの機能を提供します。
- **LFPD-ADMIN**: lfpd-admin スクリプトの機能を提供します。

LFPD コマンド

フォーマット

LFPD コマンドの構文は以下のとおりです。

```
smsg <GUEST> app lfpd (start|stop|restart|force-reload|status|startdbg) <IUCVNAME>
smsg <GUEST> app lfpd (list|startall|stopall)
```

この機能の意味は、 470 ページの『Linux on z Systems』で説明されているものと同じです。lfpd インスタンスの開始時に startdbg パラメーターを使用して診断情報を取得できます。

例

startdbg コマンドの出力にすべての lfpd 始動メッセージが表示されます。

図 *39. z/VSE VIA startdbg* コマンドの出力例

```
smsg vsevia app lfpd start fails
Ready; T=0.01/0.01 15:48:16
 15:48:19  * MSG FROM VSEVIA  : STARTING LFPD (FAILS): FAILED

smsg vsevia app lfpd startdbg fails
Ready; T=0.01/0.01 15:48:35
 15:48:38  * MSG FROM VSEVIA  : STARTING LFPD (FAILS): FAILED
15:48:38  * MSG FROM VSEVIA  : STARTUP LOG
15:48:38  * MSG FROM VSEVIA  : LOGGING STARTUP  MESSAGES TO FILE '/TMP/LFPD-CTL.2854.tmp'.
15:48:38  * MSG FROM VSEVIA  : READING CONFIGURATION FROM FILE '/ETC/OPT/IBM/VSELFPD/CONFS-
ENABLED/LFPD-FAILS.CONF'.
15:48:38  * MSG FROM VSEVIA  : IUCV_INITIALIZESOCKET: ERROR: THE BIND CALL FOR THE AFIUCV
SOCKET FAILED: ADDRESS ALREADY IN USE
15:48:38  * MSG FROM VSEVIA  : ERROR: COULD'NT INITIALIZE THE SOCKETS.
15:48:38  * MSG FROM VSEVIA  : STOPPING TO LOG THE STARTUP BECAUSE LFPD IS ABOUT TO EXIT.
```

LFPD-ADMIN コマンド

フォーマット

LFPD-ADMIN コマンドの構文は以下のとおりです。

```
smsg <GUEST> app lfpd-admin <IUCVNAME> trace start <FILENAME> (debug|packets|all)
 (single|wrap) (maxsize)
smsg <GUEST> app lfpd-admin <IUCVNAME> trace stop
smsg <GUEST> app lfpd-admin <IUCVNAME> status
```

「trace start」機能によって、指定されたファイルへの lfpd トレースが開始されます。このトレース・ファイルは CMS ミニディスク 0D4D に作成されます。FILENAME には、ファイル・タイプなしの有効な CMS ファイル名を指定する必要があります。この機能の意味は、 470 ページの『Linux on z Systems』で説明されているものと同じです。すべてのパラメーターはオプションであり、各パラメーターをどのような順序で指定してもかまいません。トレースを開始できるようにするには、ユーザーは CMS ファイル形式のディスクを装置番号 0D4D にアタッチする必要があります。トレースを停止した後にディスクをデタッチできます。テキストの lfpd トレースがバイナリー・モードで CMS ディスクに書き込まれます。z/VM 上でトレース・ファイルを読み取ることはできません。

注: 一度に開始できるトレースは 1 つのみです。

第 16 章 z/VSE Network Appliance

このセクションでは、統合 z/VSE Network Appliance (VNA) ソリューションについて説明します。

概要

z/VSE Network Appliance (VNA) は、LPAR モードで稼働している z/VSE システムに TCP/IP スタック機能を提供する、統合されたソリューションです。 z/VSE Fast Path to Linux on z Systems (LFP) 機能に基づいて構築されており、TCP/IP ソケット・アプリケーションが z/VSE 上の TCP/IP スタックを必要とせずにネットワークにアクセスすることを可能にします。

「従来」の z/VSE Linux Fast Path 機能では、Linux Fast Path デーモン (lfpd) を実行するために、ユーザーは Linux on z Systems システムのインストール、管理、および構成を行う必要がありました。この Linux on z Systems システムでは、lfpd をインストールして構成する必要があります。Linux on z Systems システムのインストールと保守を自分で行いたくないお客様には、z/VSE Linux Fast Path の使用に必要なすべてのサービスを提供する、使いやすいイメージが z/VSE Network Appliance に用意されています。

z/VSE の TCP/IP スタックに比べ、z/VSE Network Appliance は、より大量の TCP/IP トラフィック・スループットに対応し、z/VSE での処理リソースを削減します。 VNA は the IBM Secure Service Container を使用します。これは、IBM z13 サーバーおよび IBM z13s サーバーで使用可能です。 VNA は Secure Service Container LPAR にインストールできます。 VNA は、z/VSE での z/VSE Fast Path to Linux on z Systems 機能に対応するものです。 VNA では、お客様は、z/VSE Fast Path to Linux on z Systems の使用に必要なすべてのサービスを提供する、使いやすく、すぐに実行できるアプライアンスを使用できます。

最小前提条件

- IBM z13 または IBM z13s。
- LPAR モードで稼働している z/VSE 5.1 以降。
- z/VM または LPAR モードで稼働している z/VSE 6.2 以降。
- z/VSE と z/VSE Network Appliance LPAR 間の HiperSockets 接続。

IBM Resource Link にある z13 サーバーと z13s サーバーのライブラリー・ページを通じて「*IBM z Systems Secure Service Container User's Guide*」(SC28-6971) を参照できます。この資料には、SSC モード区画の構成および開始と、Secure Service Container インストーラーを使用したソフトウェア・アプライアンスのインストールに関するトピックが含まれています。 z/VSE Network Appliance には z13 または z13s のハードウェアが必要です。アプライアンスを実行する LPAR には IFL プロセッサーの使用をお勧めします。詳しくは、「*IBM z Systems Secure Service Container User's Guide*」 に説明があります。

z/VSE Network Appliance のインストール・イメージを入手するには、「Contact z/VSE」Web ページで要求してください。

https://www-03.ibm.com/systems/z/os/zvse/contact/contact.html

z/VSE Network Appliance が Secure Service Container LPAR にインストールされると、Network Appliance との以降の対話はすべて Web インターフェースを使用して行われます。インストール・プロセス中に構成した IP アドレスで Web インターフェースにアクセスします。 z/VSE Network Appliance パッケージの一部として、他にも資料が提供されています。

z/VSE Network Appliance の問題については、IBM サポートに連絡し、z/VSE の AF コンポーネントに関する PMR またはサービス要求 (SR) を開くことができます。 z/VSE Network Appliance は有効な z/VSE ライセンスと併せてサポートされます。

第 17 章 OpenSSL

このセクションでは、z/VSE に OpenSSL を実装する方法について説明します。

概要

OpenSSL は、SSL/TLS 実装と鍵管理ユーティリティーを提供するオープン・ソース・プロジェクトです。OpenSSL は C で作成されており、多くのオペレーティング・システムとハードウェア・プラットフォームで利用できます。z/VSE の OpenSSL では、独自の SSL 実装を持っていない IPv6/VSE IP スタック用の SSL が提供されます。OpenSSL の詳細については、*http://www.openssl.org/* を参照してください。

z/VSE 6.2 では、OpenSSL はシステム・コンポーネントの一部として提供されます。

```
VSE CryptoServices
5686-VS6
CLC=62S
```

OpenSSL は PRD1.BASE にインストールされ、以下のものを含みます。

- IJBSSL.phase および IJBSSLM.phase (OpenSSL 実装)
- SPEEDTST フェーズ (組み込みの OpenSSL 速度テストを呼び出します)
- NOTICES.Z (ライセンス情報)
- IJBSLVSE.OBJ (API へのアクセスを提供するものであり、アプリケーションにリンクされる必要があります)
- IJBSSL.H (関数プロトタイプを提供します)

z/VSE での固有の機能

z/OS 互換の **SSL** プログラミング・インターフェース
: この API については、「z/OS Cryptographic Services System SSL プログラミング」で説明されています。この API は、CICS Web サポート、VSE コネクター・サーバー、WebSphere® MQ for z/VSE など、z/VSE 上のすべての既存の SSL アプリケーションで使用されます。この z/OS SSL API ではネイティブの OpenSSL 関数がラップされているので、既存の z/VSE SSL アプリケーションを未変更のまま OpenSSL で実行できます。

z Systems 暗号化ハードウェアのサポート
: OpenSSL は、あらゆるキー長のすべての暗号化アルゴリズムをソフトウェア内で実行できますが、ハードウェア暗号化サポートの使用によりパフォーマンスが大幅に向上しています。また、ハードウェア・ベースの乱数生成など、ソフトウェアでは使用できないハードウェア機能を使用できます。

z/VSE で使用できない機能

z/VSE 6.2 では、以下の機能を使用できません。

- z/VSE では OpenSSL コマンド行ツールを使用できません。したがって、鍵管理はワークステーション (Windows、Linux など) 上で行われます。作成された鍵ストアは、その後、z/VSE にアップロードされます。
- z/VSE では、アルゴリズム IDEA、RC5、MDC2 を使用できません。
- RC4 は、セキュリティー問題のために、APAR DY47472 で除去されています。

ランタイム変数

z/VSE での OpenSSL の動作を制御する 5 つの変数があります。

1. パラメーター SSL$ICA を使用して、暗号ハードウェアのオンとオフを切り換えることができます。

   ```
   // SETPARM SSL$ICA = [ 'YES' | 'NO' ]
   ```

2. 変数 SSL$DBG を使用して、デバッグ・トレースを制御できます。

   ```
   // SETPARM SSL$DBG = [ 'YES' | 'NO' ]
   ```

3. デフォルトの暗号は、変数 SSL$CPH を使用してオーバーライドできます。

   ```
   // SETPARM SSL$CPH = 'cipher-string'
   ```

   この cipher-string は、 518 ページの『z/OS SSL API』 に示すのと同じ方法でコーディングされます。

   例:

   ```
   // SETPARM SSL$CPH='C027C014C013C012'
   ```

   この例では、設定可能な最高度のセキュリティーのために、ECDHE-RSA ベースの暗号のみを有効にします。

   ここで指定した場合でも、OpenSSL は特定の暗号スイートを使用しない場合があることに注意してください。 OpenSSL バージョンによっては、DES や RC4 などの弱いアルゴリズムは使用できなくなります。

4. TCP/IP for z/VSE 暗号化機能の可用性検査は、パラメーター SSL$CSI によって制御できます。

   ```
   // SETPARM SSL$CSI = [ 'YES' | 'NO' ]
   ```

   TCP/IP for z/VSE 以外の IP スタック (IPv6/VSE や Linux Fast Path など) を使用している場合、CSI 暗号化機能は使用できませんが、LIBDEF 設定によっては検査でハング状態が発生するおそれがあります。このパラメーターを「NO」に設定すると、CSI 機能に対する内部呼び出しはすべてバイパスされます。

   Encryption Facility は公開鍵ベースの暗号化に CSI 提供の RSA 関数を使用するため、CSI 提供のアルゴリズム実装の検査が Encryption Facility for z/VSE で必要になることに注意してください。

5. デフォルトのトレース宛先 (SYSLST) は、パラメーター SSL$TRC を使用して VSE ライブラリー・メンバーにリダイレクトできます。

   ```
   // SETPARM SSL$TRC = 'DD:lib.sublib(membername.membertype)'
   ```

   これは、(例えば VSE/POWER PNET SSL/TLS で) SYSLST を使用できない場合に役立ちます。

以下のセクションでは、次のメイン・トピックを扱います。

- 『鍵ストアの考慮事項』
- 492 ページの『OpenSSL 用の z/VSE アプリケーションのプログラミング』
- 517 ページの『OpenSSL 速度テストの実行』

鍵ストアの考慮事項

SSL アプリケーションを実行するには、RSA 公開鍵/秘密鍵のペアと SSL 証明書が含まれる鍵ストアが必要です。

さまざまなタイプの鍵ストアが存在します。z/VSE に関連する鍵ストアは以下のとおりです。

PFX RSA Security によって最初に定義された Personal Information Exchange (PFX) 形式は PKCS#12 標準に準拠します。PFX ファイルには複数の鍵と証明書を含めることができます。また、PFX ファイル自体はパスワードで保護されます。PFX ファイルは、Microsoft Internet Explorer、Mozilla Firefox など、各種 Web ブラウザーでサポートされます。PFX 形式では、ファイル拡張子 p12 も使用されます。

JKS Oracle が提供する Java Key Store (JKS) 形式は、Java アプリケーション用の標準の鍵ストア形式です。JKS ファイルはパスワードで保護されます。通常、JKS ファイルを Web ブラウザーで処理することはできません。`keytool.exe` は、各 Java インストールの一部であり、JKS 鍵ストアの維持にも使用できます。

VSE 鍵リング・メンバー

Connectivity Systems International が発明したこのタイプの鍵ストアは、3 つの VSE ライブラリー・メンバーで構成されます。これらの VSE ライブラリー・メンバーのメンバー名は同じですが、メンバー・タイプは異なります。

- PRVK (RSA 鍵ペアが含まれます)
- ROOT (SSL ルート証明書が含まれます)
- CERT (SSL サーバー証明書が含まれます)

他の形式ではすべての鍵と証明書が 1 つのファイルに格納されますが、CSI 鍵リング形式ではアイテムごとに 1 つの独立した VSE ライブラリー・メンバーが使用されます。

RSA 鍵ペアと SSL 証明書の作成など、z/VSE 固有の公開鍵インフラストラクチャー (PKI) は、Keyman/VSE を使用して管理できます。

PEM Privacy-enhanced mail (PEM) 形式は OpenSSL で使用されます。PEM ファイルには、RSA 鍵ペア、SSL 証明書、またはその両方を含めることができます。パスワードで保護することはできますが、パスワードで保護してはなりません。他の鍵ストア形式は単なるバイナリーですが、PEM ファイルの内容はベース 64 でエンコードされています。

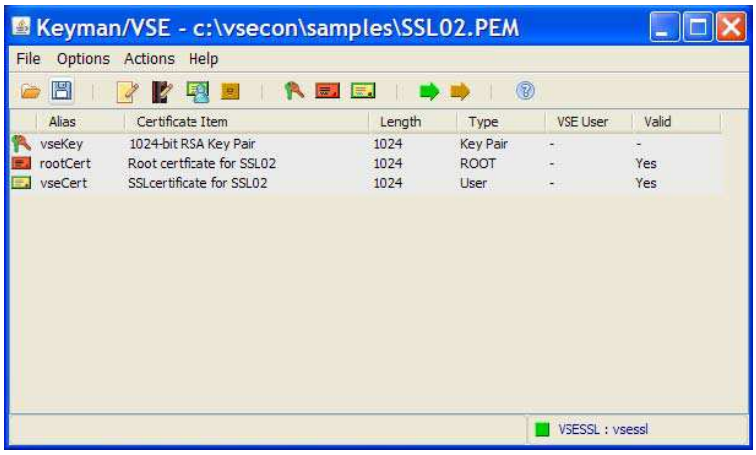
Keyman/VSE を使用した鍵ストアの作成

最新バージョンの Keyman/VSE (ビルド日付が 2013 年 1 月以降) を使用すると、PEM ファイルを容易に作成して z/VSE にアップロードできます。

テスト環境では自己署名証明書の使用が必要になる場合がありますが、実稼働環境では認証局 (CA) によって発行された証明書を使用する必要があります。

以下の手順は、Keyman/VSE を使用して PEM ファイルを作成し、アップロードする方法を示しています。

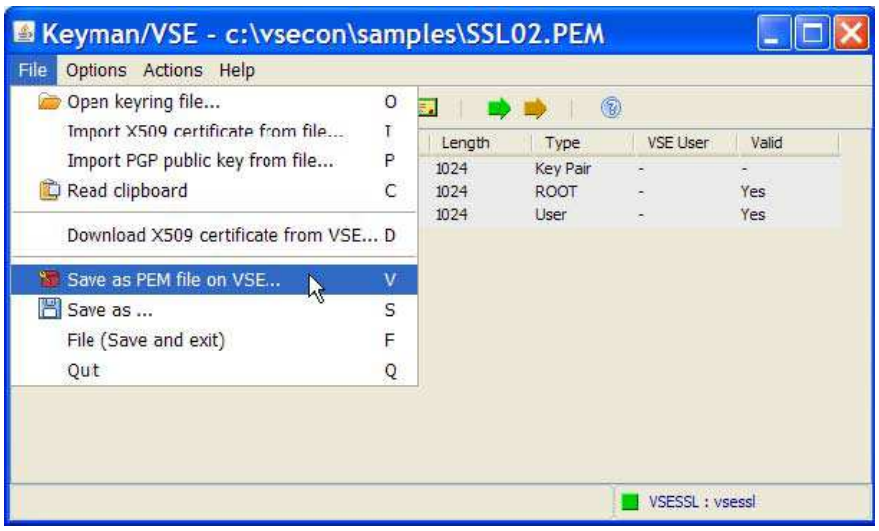
1. RSA 鍵、SSL、および SSL ルート証明書を作成します。



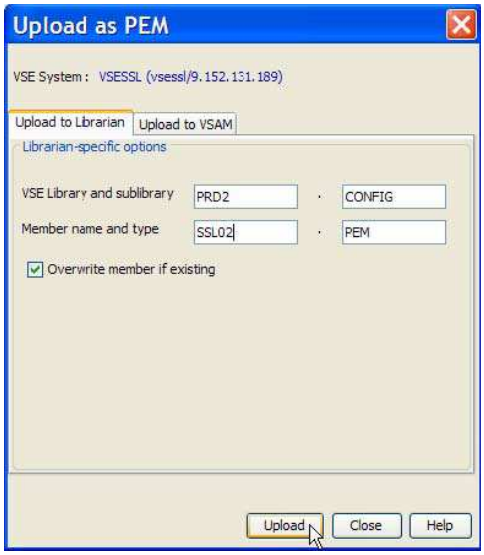
2. 「ローカル・プロパティー (Local Properties)」を開き、PEM ファイル名を指定します。

PEM パスワードの指定は現在サポートされていません。IPv6/VSE が、パスワードで保護された PEM ファイルを現在サポートしていないためです。

3. PEM ファイルを保存して z/VSE にアップロードします。



「PEM としてアップロード (Upload as PEM)」ポップアップ・ウィンドウが表示されました。



4. 現在、OpenSSL アプリケーションが構成されています。

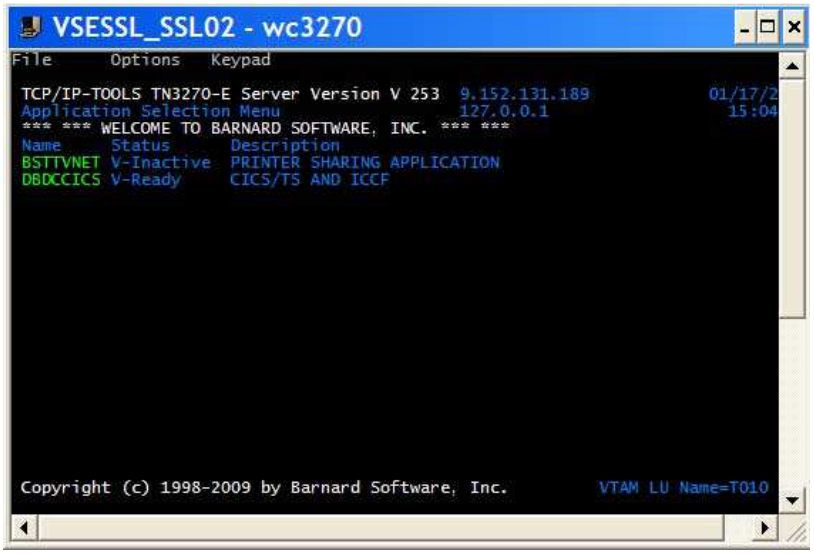
   現在、IPv6/VSE が z/VSE の唯一の OpenSSL アプリケーションです。以下の手順では、OpenSSL を使用するように IPv6/VSE を構成する方法の概要を示します。 詳しくは、「IPv6/VSE Installation Guide」、「Programming Guide」および「User's Guide」を参照してください。

5. BSTTATLS の構成

```
KEYRING CRYPTO.KEYRING
*
KEYFILE SSL02
SECTYPE TLSV1
OPTION SERVER
ATTLS 23 AS 992 SSL
```

6. IPv6/VSE で提供されている BSTTATLS および Telnet サーバー BSTTVNET を開始します。
7. 必要であれば、クライアントをセットアップします。

8. 接続します。



OpenSSL 用の z/VSE アプリケーションのプログラミング

2 つの方法を使用して、OpenSSL を使用する z/VSE アプリケーションを作成できます。

ネイティブの **OpenSSL API** の使用

OpenSSL Web サイトで説明されているこの OpenSSL API は、数百個の関数で構成されます。z/VSE では、この API のサブセットは IJBSLVSE.OBJ によって提供されます。 IJBSSL フェーズで関数にアクセスするには、IJBSLVSE.OBJ をアプリケーションにリンクします。 この API は、LE/C ランタイムの使用を必要とします。

z/OS SSL API の使用

z/VSE で OpenSSL を使用する方法として、この方法をお勧めします。API については、「z/OS Cryptographic Services System SSL プログラミング」で説明されています。C データ構造体は、組み込みファイル `gskssl.h` で定義されています。この組み込みファイルは LE/C の一部であり、PRD2.SCEEBASE で出荷されます。

この API は、現在、z/OS では非推奨となっており、新しい z/OS アプリケーションではこれ以上使用すべきではありませんが、これは、依然として z/VSE 用の実際の SSL API です。TCP/IP for z/VSE ではまさしくこの API を提供しているため、現在、既存のすべての VSE SSL アプリケーションで、この API が使用されており、OpenSSL を未変更のまま使用できます。 GSK 関数の API の詳細については、 518 ページの『z/OS SSL API』を参照してください。

EZASOCKET または **EZASMI** のプログラミング・インターフェースの使用

OpenSSL 実装を C 以外のプログラミング言語から使用することもできます。 OpenSSL 実装を選択するには、EZA マルチプレクサーを構成する必要があります。詳しくは、 95 ページの『第 10 章 使用する TCP/IP と SSL の実装を選択』を参照してください。

以下のセクションで、z/OS SSL API の使用方法と 2 つの API の切り替え方法の詳細について説明します。

組み込みファイル

SSL 関連の 2 つの組み込みファイルが z/VSE とともに提供されます。

- **sslvse.h** - TCP/IP lib.sublib の CSI SSL 実装と一緒に出荷されます。
- **gskssl.h** - PRD2.SCEEBASE の LE/C ソケット・インターフェースと一緒に出荷されます。

sslvse.h 内の関数プロトタイプが #pragma リンケージ OS を使用して定義されるため、gskssl.h に対して OpenSSL アプリケーションだけでなく OpenSSL もコンパイルする必要があります。これにより、gsk 関数へのパラメーターの引き渡しでエラーが発生します。

gskssl.h が使用されている場合、gsk API 関数は、z/OS API で説明されている戻りコードを使用します。この戻りコードは、CSI SSL 実装で現在使用されているものとは異なります。例えば、VSE コネクター・サーバーは gskssl.h に対してコンパイルされますが、CSI 組み込みファイルを使用するカスタマー・アプリケーションが存在する場合があります。

渡されるソケット番号

gsk アプリケーションによって渡されるソケット番号 (int s) は単なるソケット番号とは限りません。

CWS、VSE コネクター・サーバー、および MQ のような VSE アプリケーションは、プライベート構造体にポインターを渡します。この構造体内のどこかにソケットが格納されています。

```
typedef struct _mysock {
  int somefield;
  char* someptr;
  int socketno;
} MYSOCK;

#pragma linkage (skread,OS)
int skread(int s, void * data, int len)
{
  int rc;
  MYSOCK* mys = (MYSOCK*)s;
  rc = recv(mys->socketno, data, len, 0);
  return(rc);
}
```

注: アプリケーション・フェーズが独立したフェーズである場合にのみ、正しいパラメーターを渡すために #pragma ステートメントを使用する必要があります。アプリケーションを OpenSSL コード (OpenSSL オブジェクト) とともにリンクする場合は、#pragma ステートメントを使用しないでください。

次に、アプリケーションは以下のようにして skread および skwrite の関数ポインターを sk_soc_init_data 構造体に指定します。

```
typedef struct _gsk_soc_init_data { /* Secure soc init data */
  int fd; /* file descriptor */
  gsk_handshake hs_type; /* client or server handshake */
  char * DName; /* keyring entry Distinguished */
```

```
  /* name. When NULL the default */
  /* keyring entry is used */
  char * sec_type; /* Type of application */
  /* 0 CLIENT */
  /* 1 SERVER */
  /* 2 SERVER_WITH_CLIENT_AUTH */
  /* 3 CLIENT_NO_AUTH */
  char * cipher_specs; /* SSLV2 not used by VSE */
  char * v3cipher_specs; /* SSLV3 cipher suites */
  int (* skread) /* User supplied READ routine */
  (int fd, void * buffer, int num_bytes);
  int (* skwrite) /* User supplied WRITE routine */
  (int fd, void * buffer, int num_bytes);
  unsigned char cipherSelected[3]; /* SSLV2 not used by VSE */
  unsigned char v3cipherSelected[2]; /* Cipher Spec used */
  int failureReasonCode; /* failure reason code */
  gsk_cert_info * cert_info; /* This information is read from*/
  /* from the client certificate */
  /* when client authentication is*/
  /* enabled */
  gsk_init_data * gsk_data; /* Pointer to init data */
} gsk_soc_init_data;
```

関数ポインターを指定する場合、fetchep を使用する必要があります。

```
typedef void (*FETCH_PTR)(int);
init_data.skread = (int (*) (int,void*,int))fetchep((FETCH_PTR)skread);
init_data.skwrite = (int (*) (int,void*,int))fetchep((FETCH_PTR)skwrite);
```

この理由は、skread および skwrite 関数はアプリケーションによって実装されますが、フェーズ IJBSSL によって呼び出されるためです。

コールバック・ルーチン

GSK API を使用する場合、ソケット上でのデータ送信に使用される読み取りルーチンと書き込みルーチンが呼び出し元で指定されます。

これは、SSL コンポーネントがソケットに直接的にアクセスしないことを意味します。また、GSK API が他のソケット呼び出し (givesocket、takesocket、ctrl など) を使用できないことも意味します。しかし、OpenSSL がソケット呼び出しを行います。低レベルのソケット関数は eos.h で定義されます。ここでは、VSE 用に以下の #define が使用されています。

```
#define readsocket(s,b,n) read((s),(b),(n))
#define writesocket(s,b,n) write((s),(b),(n))
```

これにより、OpenSSL はアプリケーションで指定された読み取り/書き込みルーチン (skread/skwrite) を使用します。GSK API で考慮されているのはこの 2 つのルーチンのみのため、現在、解決策または他のソケット呼び出しはありません。また、アプリケーションによって渡されるソケット番号 (int s) は単なるソケット番号とは限らないことに注意する必要があります。CWS、VSE コネクター・サーバー、MQ のような VSE アプリケーションは、プライベート構造体にポインターを渡します。この構造体内のどこかにソケットが格納されています。柔軟性を高めるには、以下の例のようなコードを書いて、ソケット番号をデータ構造体に入れることができます。

```
typedef struct _mysock {
  int somefield;
  char* someptr;
  int socketno;
```

```
} MYSOCK;

/* I/O routine to perform a read function for SSL VSE */
#pragma linkage (skread,OS)
int skread(int fd, void * data, int len)
{
  int rc;
  MYSOCK* mys = (MYSOCK*)fd;
  rc = recv(mys->socketno, data, len, 0);
  return(rc);
}

/* I/O routine to perform a write function for SSL VSE */
#pragma linkage (skwrite,OS)
int skwrite (int fd, void * data, int len)
{
  int rc;
  MYSOCK* mys = (MYSOCK*)fd;
  rc = send(mys->socketno, data, len, 0);
  return(rc);
}
```

gsk_secure_soc_init() が呼び出される前に、2 つのコールバック・ルーチン (skread() と skwrite()) のアドレスが gsk_soc_init_data 構造体に指定されています。

ソケット呼び出し

gsk-API を使用する場合、ソケット上でのデータ送信に使用される読み取りおよび書き込みのコールバック・ルーチンが呼び出し元で指定されます。

SSL コンポーネントはソケットに直接アクセスしません。また、gsk-API は他のソケット呼び出し (givesocket、takesocket、ctrl など) を使用できません。

ただし、OpenSSL がソケット呼び出しを行います。低レベルのソケット関数は eos.h で定義されます。ここでは、VSE 用に以下の #defines が使用されています。

```
#define get_last_socket_error() errno
#define clear_socket_error() errno=0
#define ioctlsocket(a,b,c) ioctl(a,b,c)
#define closesocket(s) close(s)
#define readsocket(s,b,n) read((s),(b),(n))
#define writesocket(s,b,n) write((s),(b),(n))
```

これにより、OpenSSL はアプリケーションで指定された読み取り/書き込みルーチン (skread/skwrite) を使用します。現在、gsk-API ではこの 2 つのルーチンのみが考慮されているため、他のソケット呼び出しはサポートされません。gsk-API 実装では vse_readsocket および vse_writesocket ルーチンが実装されており、アプリケーションで指定された読み取り/書き込みルーチンが呼び出されます。

```
int (*skread)(int s, void* b, int n);
int (*skwrite)(int s, void* b, int n);

int vse_readsocket(int s, void* b, int n)
{
  return skread(s, b, n);
}
```

```
int vse_writesocket(int s, void* b, int n)
{
  return skwrite(s, b, n);
}
```

gsk と OpenSSL のソケット呼び出しの切り替え

アプリケーションで gsk API とネイティブの OpenSSL API を動的に使用できるように、gsk_initialize() は、フェーズ IJBSSL でグローバル変数 ssl_use_gsk_callbacks を設定します。

これにより、OpenSSL コードで gsk コールバックが使用されます。ソケット呼び出しを行う OpenSSL モジュールは 2 つのみです (bssconn.c および bsssock.c)。VSE 用に以下の変更が加えられています。

```
if (ssl_use_gsk_callbacks)
  ret=vse_readsocket(b->num,out,outl);
else
  ret=readsocket(b->num,out,outl);
```

グローバル変数は gsk_uninitialize() でリセットされます。

以下のセクションでは、鍵リング・ファイルの名前と場所の指定方法や、アプリケーションで使用する SSL 暗号のリストなど、z/VSE 固有の側面をいくつか説明します。

鍵リングの指定

gsk アプリケーションで鍵リングを指定する方法は 2 つあります。

これは、PEM ファイルを VSE ライブラリー・メンバーまたは VSAM ファイルのどちらとしてアップロードしたかによって異なります。どちらの場合も、鍵リングの場所は gsk_initialize() 関数が呼び出されるときに指定され、鍵リングの名前は gsk_secure_soc_init() 関数が呼び出されるときに指定されます。

lib.sublib に空ストリングが指定された場合、gsk ラッパーは鍵リング・ラベル (DName) が VSAM ファイル名であると想定します。lib と sublib に有効な値が指定された場合、鍵リングのラベルは VSE ライブラリー・メンバー名として処理されます。この場合、メンバー・タイプは PEM です。

鍵リング・タイプ、ライブラリアン

ライブラリアン・タイプの鍵リングでは、VSE ライブラリーとサブライブラリーが gsk_initialize() 呼び出しで指定されます。

```
char * keyring = "CRYPTO.KEYRING";
gsk_init_data init_data;
...
init_data.keyring = keyring;
rc = gsk_initialize(&init_data);
```

PEM ファイルのメンバー名は、DName パラメーターを使用して、gsk_secure_soc_init() 呼び出しで指定されます。メンバー・タイプは常に PEM である必要があります。

```
gsk_soc_data * socdata;
gsk_soc_init_data sock_init_data;
...
sock_init_data.DName = "MYKEY"; // VSE library member name
socdata = gsk_secure_soc_init(&sock_init_data);
```

鍵リング・タイプ、VSAM

VSAM タイプの鍵リングでは、gsk_initialize() 呼び出しのパラメーター「keyring」に空ストリングが指定されます。

```
char * keyring = "";
gsk_init_data init_data;
...
init_data.keyring = keyring;
rc = gsk_initialize(&init_data);
```

gsk_secure_soc_init() 呼び出しで定義されている鍵リング名 (DName) は VSAM ファイルのラベルを示します。

```
gsk_soc_data * socdata;
gsk_soc_init_data sock_init_data;
...
sock_init_data.DName = "MYKEY"; // VSAM file label
socdata = gsk_secure_soc_init(&sock_init_data);
```

パスワードで保護された鍵リングの使用

パスワードで保護された鍵リングを PEM ファイルで使用する場合、鍵ストアにアクセスするための PEM パスフレーズを各クライアントまたはサーバーで指定する必要があります。

gsk-API を使用する場合、gsk_initialize() を呼び出すときに、鍵リングのパスワードを gsk_init_data 構造体に指定します。

```
char * keyring = "";
gsk_init_data init_data;
...
init_data.keyring = keyring;
init_data.keyring_pw = "ssltest";
rc = gsk_initialize(&init_data);
```

注: gsk-API ではパスワードで保護された PEM ファイルがサポートされますが、現在、IPv6/VSE ではパスワードで保護された PEM ファイルはサポートされません。

z/VSE 上の OpenSSL は、以下を想定しています。

1. PEM ファイルが ASCII プラットフォームで作成されたため、パスワードは ASCII でエンコードされています。
2. z/VSE で実行されている gsk アプリケーションでは、パスワードが EBCDIC で指定されます。

したがって、関連する OpenSSL 関数を呼び出して PEM ファイルを開く前に、パスワードを ASCII に変換します。

注: パスワードの文字によっては、EBCDIC から ASCII への変換で問題が発生する場合があります。現在、コード・ページを指定することはできません。

サポートされる暗号スイート

OpenSSL では、以前に TCP/IP for z/VSE で使用できていた暗号スイートよりも多くの暗号スイートを使用できます。

OpenSSL では、暗号スイートのリストを指定できるのに加えて、DEFAULT、ALL、HIGH など、キーワードを使用できます。ただし、gsk API を使用するアプリケーションでは、以前に CSI 実装を使用するときに行っていたように、16 進コードのリストを指定する必要があります。その後、16 進コードのリストは、OpenSSL で読み取ることができる暗号スイートのリストに内部的に変換されます。

次の表に、OpenSSL とともに使用できる暗号スイートのリストを示します。表には、SSL ハンドシェーク用の RSA アルゴリズムを使用する暗号スイートのみが示されています。

表 *11.* 現在サポートされている *OpenSSL* 暗号スイート

16 進コード	**OpenSSL** 表記	参照する注の番号
0A	DES-CBC3-SHA	
2F 35	AES128-SHA AES256-SHA	
3C 3D	AES128-SHA256 AES256-SHA256	1
16 33 39 67 6B	EDH-RSA-DES-CBC3-SHA DHE-RSA-AES128-SHA DHE-RSA-AES256-SHA DHE-RSA-AES128-SHA256 DHE-RSA-AES256-SHA256	2
C011 C012 C013 C014 C027	ECDHE-RSA-RC4-SHA ECDHE-RSA-DES-CBC3-SHA ECDHE-RSA-AES128-SHA ECDHE-RSA-AES256-SHA ECDHE-RSA-AES128-SHA256	3

1. これらの暗号スイートは TLSv1.2 に属しており、OpenSSL 1.0.1e 以上が必要です。
2. VSE がサーバーである場合は、これらの暗号スイートには DH パラメーターの設定が必要です。
3. VSE がサーバーである場合は、これらの暗号スイートには DH パラメーターと EC 鍵の設定が必要です。

注: 新しい OpenSSL バージョンでは、既知のセキュリティー問題のために、これ以上の暗号スイートがサポートされなくなる場合があります。最新の情報については、https://www.openssl.org/ で公開されている OpenSSL 資料を確認してください。

488 ページの『ランタイム変数』に示すように、暗号スイートのこのリストは、SSL$CPH 変数でオーバーライドできます。

LE/C アプリケーションの場合、LE/C マルチプレクサーから TCP/IP for z/VSE で OpenSSL を使用できます。詳しくは、ICCF ライブラリー 62 のスケルトン EDCTCPMC を参照してください。

OpenSSL 暗号スイートの詳細については、http://www.openssl.org/docs/apps/ciphers.html を参照してください。暗号スイートの詳細については、RFC 2246 および 3268 を参照してください。

TCP/IP for z/VSE でサポートされる暗号スイート

表 12 は、現在 TCP/IP for z/VSE でサポートされている SSL 暗号スイートを示しています。この表では、*VSE* コネクター・クライアント 用にこれらの暗号スイートを定義するときに使用するフォーマットを示しています。

VSE コネクター・サーバー 用にこれらの暗号スイートを定義するときに使用するフォーマットについては、「*IBM z/VSE* 管理」を参照してください。

表 *12.* 現在 *TCP/IP for z/VSE* でサポートされている *SSL* 暗号スイート

16 進コード	暗号スイート	参照する注の番号
01	SSL_RSA_WITH_NULL_MD5	1, 2
02	SSL_RSA_WITH_NULL_SHA	1, 2
	SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	1, 2
	SSL_RSA_WITH_DES_CBC_SHA	1
0A	SSL_RSA_WITH_3DES_EDE_CBC_SHA	1
2F	TLS_RSA_WITH_AES_128_CBC_SHA	1
35	TLS_RSA_WITH_AES_256_CBC_SHA	1

注**:**

1. 暗号スイート `NULL_MD5` (X'01')、`NULL_SHA` (X'02')、および `RSA1024_EXPORT_DESCBC_SHA` (X'62') では、SSL 3.0 ハンドシェークが必要です。 TLS 1.0 ハンドシェークは使用できません。
2. 暗号スイート `SSL_RSA_EXPORT_WITH_DES40_CBC_SHA` (X'08') 、`SSL_RSA_WITH_DES_CBC_SHA` (X'09')、および `SSL_RSA_WITH_3DES_EDE_CBC_SHA` (X'0A') は、SSL 3.0 と TLS 1.0 の両方のハンドシェークで使用できます。

2048 ビットおよび 4096 ビットの SSL/TLS ハンドシェークではさらに、Crypto Express カードと最新バージョンの TCP/IP for z/VSE が必要です。詳しくは、「*IBM z/VSE* 計画」の『z/VSE 6.2 ハードウェア・サポート』を参照してください。

暗号スイートの指定

gsk クライアントと gsk サーバーは、gsk API の要求に従って、暗号スイートのリストを指定します。

```
char * ciphers = "2F350A";
...
sock_init_data.v3cipher_specs = ciphers;
...
socdata = gsk_secure_soc_init(&sock_init_data);
```

次に、このリストは、OpenSSL で読み取ることができるフォームに変換されます。この例では、OpenSSL に内部的に渡されるストリングは次のようになります。

```
AES128-SHA:AES256-SHA:DES-CBC3-SHA:DES-CBC-SHA
```

サポートされる RSA 鍵の長さ

z/VSE では、RSA 鍵の長さは 4096 ビットに制限されます。これが、現在、暗号カードでサポートされる上限であるためです。

OpenSSL 自体は最大で 16384 ビットの RSA キーを処理できます。z/VSE では、組み込みファイル `rsa.h` に次の変更が加えられています。

```
#ifndef OPENSSL_RSA_MAX_MODULUS_BITS
//# define OPENSSL_RSA_MAX_MODULUS_BITS 16384
# define OPENSSL_RSA_MAX_MODULUS_BITS 4096
#endif
```

ソフトウェア・ベースの暗号化を使用するのに加えて、この制限を除去しても意味はありません。暗号化ハードウェアを使用する場合と比較して、ソフトウェア・パフォーマンスは非常に限られているためです。

デバッグ

OpenSSL にはさまざまな組み込みデバッグ機能があります。

gsk API を使用する場合

z/OS gsk API でデバッグ・トレースのオンとオフを切り替えるには、次のように JCL 変数を使用できます。

```
SSL$DBG = [YES | NO]
```

変数は **gsk_initialize()** 関数内で評価されます。したがって、変数は z/OS gsk インターフェースでのみ場合のみ使用できます。

両方の API を使用する場合

ソース・モジュールを再コンパイルしないでデバッグのオンとオフができるように、z/VSE では、モジュール IJBSLVSE 経由で次の 2 つの関数が提供されます。

```
extern int debug=0;
void ssl_enable_debug(void);
void ssl_disable_debug(void);
```

デバッグを有効にすると、静的グローバル変数「debug」が 1 に設定されます。デバッグを無効にすると、値はゼロにリセットされます。この 2 つの関数は、あらゆる z/VSE gsk アプリケーションまたは OpenSSL アプリケーションで使用できます。

デバッグが有効にされている場合、フェーズ IJBSSL からのデバッグ出力は SYSLST に表示されます。次のように、アプリケーションを呼び出すときに、PARM ストリングに DEBUG パラメーターを指定して gsk アプリケーションまたは OpenSSL アプリケーションのデバッグを有効にしたとします。

```
// EXEC MYAPP,PARM='DEBUG'
```

アプリケーション・コードはパラメーターを読み取って、関連する関数を呼び出し、デバッグを有効または無効にします。

ハードウェア暗号化サポート

ハードウェア暗号化サポートは既存の z/VSE SSL アプリケーションに透過的に提供されます。これらのアプリケーションでは、z/OS gsk API が使用されるためです。

アプリケーションで使用される API によって、以下のいずれかが適用されます。

gsk API を使用する場合

z/OS gsk API でハードウェア暗号化サポートのオンとオフを切り替えるには、次の JCL 変数を使用できます。

```
SSL$ICA = [YES | NO]
```

変数は **gsk_initialize()** 関数内で評価されます。したがって、変数は z/OS gsk インターフェースでのみ場合のみ使用できます。

両方の API を使用する場合

ソース・モジュールを再コンパイルしないでデバッグのオンとオフができるように、z/VSE では、モジュール IJBSLVSE および IJBSLACC 経由で次の 2 つの関数が提供されます。

```
extern int ssl_use_ibmca=1;
void ssl_enable_ibmca(void);
void ssl_disable_ibmca(void);
```

ハードウェア暗号化サポートを有効にすると、静的グローバル変数「ssl_use_ibmca」が 1 に設定されます。**ssl_disable_ibmca()** を呼び出すと、値はゼロにリセットされます。この 2 つの関数は、あらゆる z/VSE gsk アプリケーションまたは OpenSSL アプリケーションで使用できます。

アプリケーションを呼び出すときに PARM ストリングにパラメーター IBMCA を指定します。

```
// EXEC MYAPP,PARM='IBMCA'
```

アプリケーション・コードはパラメーターを読み取って、関連する関数を呼び出し、ハードウェア暗号化サポートを有効または無効にします。

Transport Layer Security - TLSv1.2

3 現在、TLSv1.2 は SSL プロトコルの最新バージョンです。これは、SHA-1 関数の代わりに SHA-256 ハッシュ・アルゴリズムを使用する新規 SSL/TLS 暗号スイートを導入しており、データ保全性を大幅に強化します。

TLSv1.2 について詳しくは、以下の Internet Engineering Task Force の Web サイトを参照してください。

```
http://tools.ietf.org/html/rfc5246
```

以下の SSL 暗号スイートとその関連 16 進値を使用できます。

```
0x3B TLS_RSA_WITH_NULL_SHA256
0x3C TLS_RSA_WITH_AES_128_CBC_SHA256
0x3D TLS_RSA_WITH_AES_256_CBC_SHA256
```

TLSv1.2 は、OpenSSL 1.0.1e 以降でサポートされます。 z/VSE は、以下の理由から、TLSv1.2 を必要とします。

- NIST Special Publication 800-131A (2011 年 1 月付) では、2013 年 12 月 31 日以降、SHA-1 ハッシュ関数の使用は、非デジタル・シグニチャー・アプリケーションを対象とする場合を除き、許可されないことが言明されています。
- IBM Global Security ポリシーにより、すべての IBM 製品は NIST Special Publication 800-131 に準拠することが求められます。

TLSv1.2 を IPv6/VSE 製品とともにセットアップして使用する方法の詳細および例については、IBM Redbook 「Enhanced Networking on IBM z/VSE」を参照してください。

Diffie-Hellmann の概要

Diffie-Hellman (DH) 鍵協定方式は、RSA を使用する SSL ハンドシェーク・プロセス中の、従来の方法による暗号化鍵ネゴシエーションに代わるものです。

Diffie-Hellman は認証を提供しないため、例えば RSA などの、追加の認証メカニズムと一緒に使用されます。 Diffie-Hellman については、以下の Internet Engineering Task Force の Web サイトにある、RFC 2631 で説明されています。

`http://www.ietf.org/rfc/rfc2631.txt`

これらの考慮事項では、通信パートナーの認証方法ではなく、SSL セッション鍵の作成方法および交換方法に関して、DH と RSA とを比較しています。

- RSA と比べて DH を使用することの主な利点は、セッション鍵がネットワークを介して送信されないため、「Perfect Forward Secrecy (PFS)」が提供できるということです。 PFS を使用すると、RSA 秘密鍵が漏えいまたは破損した可能性がある場合に、記録された SSL セッションを後で暗号化解除することができません。
- DH の主な欠点は、CPU 消費量が増えることです。 DH を使用して SSL セッションを確立すると、RSA を使用する場合に比べ、CPU の消費量は約 30% 増えます。

次のセクションでは、RSA を使用したセッション鍵の交換方法について説明し、DH と RSA の相違点を示します。

RSA

セッション鍵を交換する 1 つの方法は、それを RSA 公開鍵で保護することです。

これは TCP/IP for z/VSE 上での SSL の動作ですが、OpenSSL をベースとする IPv6/VSE でも動作します。 503 ページの図 40 は、RSA を使用した SSL セッション鍵交換を示しています。 **1** まずクライアントがサーバーにアクセスします。 **2** サーバーは、サーバーの秘密 RSA 鍵で署名されているデジタル SSL 証明書でラッピングされた、公開 RSA 鍵を送信します。 **3** クライアントは、認証局 (CA) の支援を受けて、サーバー署名を検査できます。 **4** クライアントは、ラ

ンダム・セッション鍵を作成し、サーバーの公開鍵を使用して暗号化し、サーバーに送信します。これにより、セッション鍵は SSL セッション・データの一部となりますが、これが弱点となります。 5 最後に、サーバーはこのセッション鍵の使用を確認します。その後、双方は暗号化データの転送を開始できます。

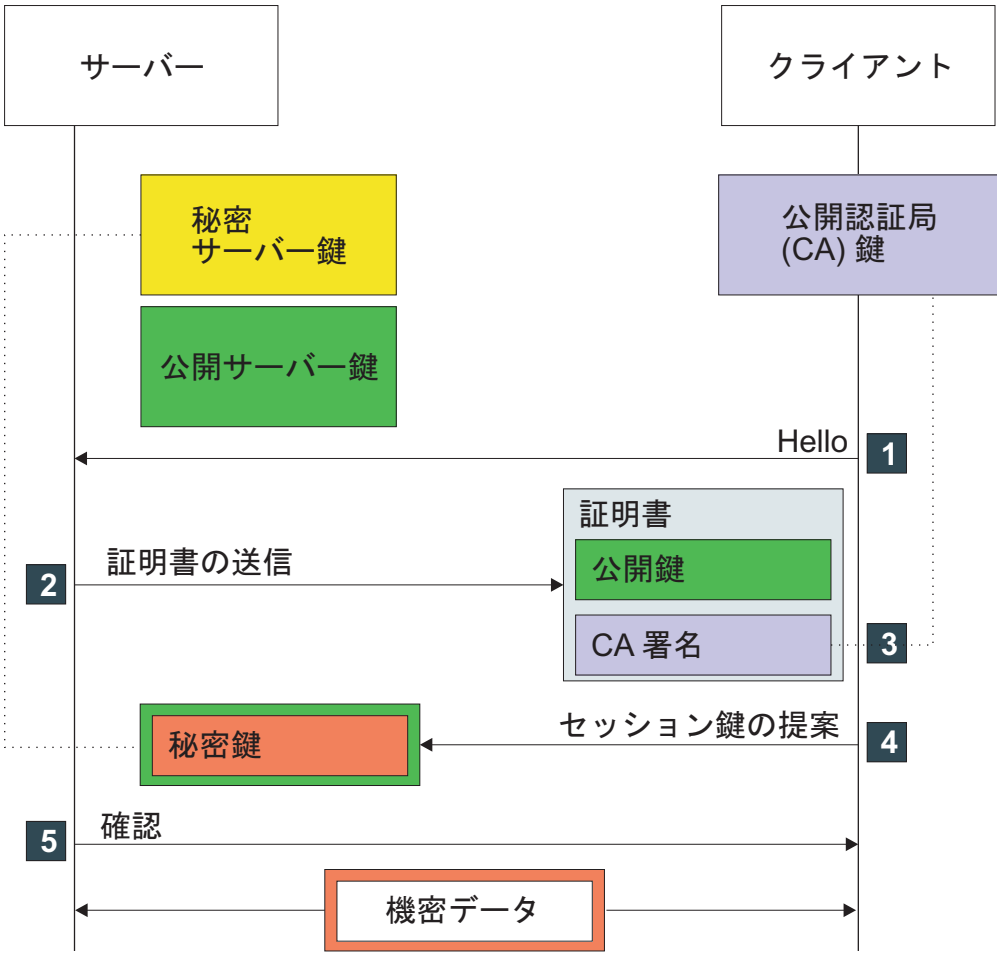


図 40. *RSA* を使用した *SSL* セッション鍵交換

弱点は、秘密セッションが SSL セッション・データの一部になるということです。鍵は、公開サーバー RSA 鍵により暗号化されて、クライアントからサーバーに送信されます。専用サーバーの RSA 秘密鍵で暗号漏えい、盗難、または破損が発生した場合、セッション鍵は安全ではなくなり、セッションが記録されて保管されていれば、セッション・データ全体を暗号化解除して読み取ることが可能になってしまいます。

Diffie-Hellman

セッション鍵のもう 1 つの交換方法は、Diffie-Hellman を使用することです。Diffie-Hellman を使用すると、セッション鍵はネットワークを介して送信されないため、ネットワーク・セッション・データの一部になることはありません。

図 41 は、セッション鍵が DH を使用してネゴシエーションされる方法を示しています。この方法により、一般的に使用される表現は「セッション鍵の交換」ではなく、DH 鍵協定プロセスによる「共通セッション鍵での合意」になります。

1 クライアントがサーバーにアクセスします。サーバーはその Diffie-Hellman パラメーターをクライアントに送信します。この DH パラメーターは、大きい素数 (*p*) と、いわゆるジェネレーター (*g*) とで構成され、$0 < g < p$ となります。OpenSSL の場合、*g* は必ず 2 になります。 **2** 両方の通信パートナーは、{1 ... *p*-2} の範囲内で乱数を生成します。これら 2 つの乱数は秘密にされ、回線上では送信されません。それらは DH 秘密鍵と呼ばれます。 **3** 両者は特定の計算を実行し、保護されていない媒体で交換される 2 つの値 *A* と *B* を導出します。 **4** これで両者は同じセッション鍵を導出できます。

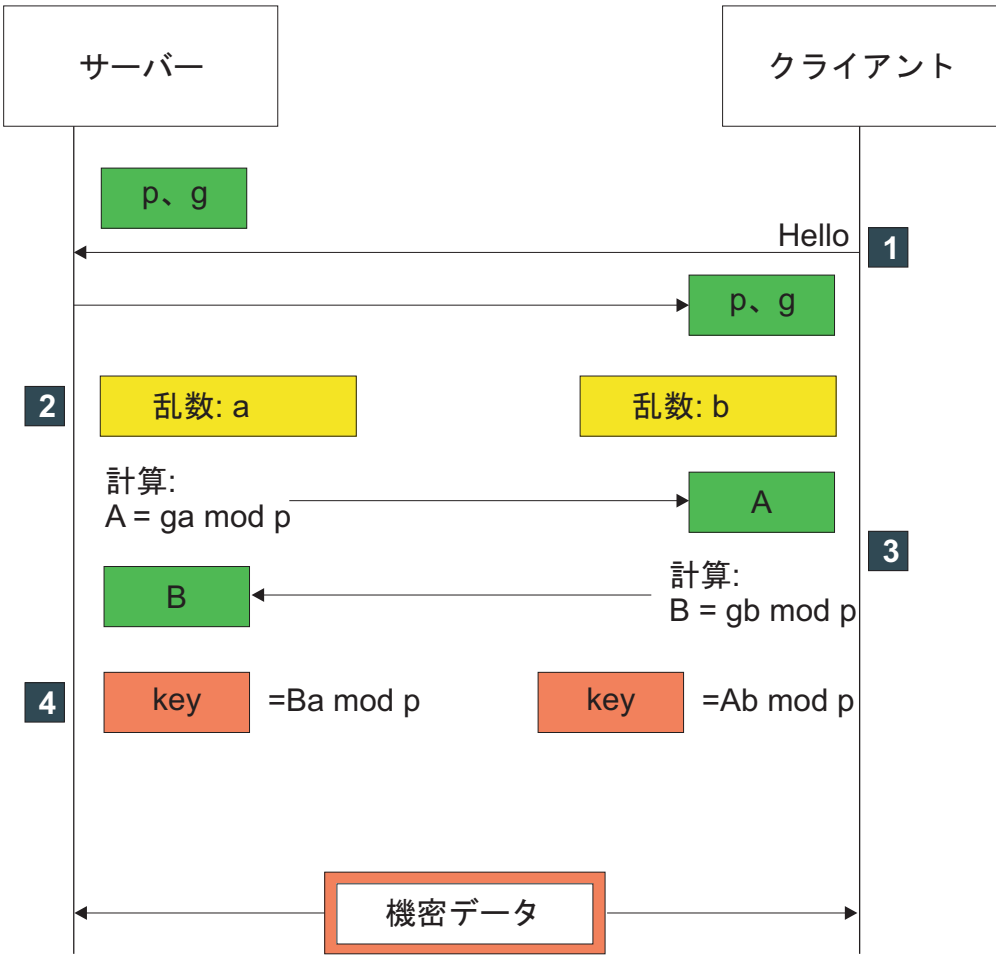


図 *41. Diffie-Hellman* を使用した *SSL* セッション鍵交換

重要な点は、暗号鍵が両方の側で、それぞれ独立して作成されるということです。これにより、後で特定の記録されたネットワーク・セッションからセッション鍵を割り出すことは不可能になります。簡単に言えば、図 41 は 2 つの通信パートナーの認証方法を示さないということです。実際には、DH はたいていの場合、RSA を使用した認証と一緒に使用されます。

Diffie-Hellman のバリアント

SSL/TLS には、Diffie-Hellman の 3 つの主要なバリアントがあります。

匿名モード
: 匿名モードは認証を使用しないため、中間者攻撃には脆弱です。匿名の Diffie-Hellman は使用すべきではありません。

静的モード
: 静的 Diffie-Hellman は、すべての接続に対して、2 つの DH 秘密鍵の少なくとも 1 つを、変更しないまま再利用します。両方の側が DH 秘密鍵を再利用するモードは、「*Static-Static*」という用語で表現されます。一方の側のみが同じ鍵を使用するモードは、「*Ephemeral-Static*」という用語で表現されます。実装によっては、パフォーマンス上の理由から、特にサーバー側で単一の静的 DH 秘密鍵を持つことが妥当な場合があります。

一時 **(Ephemeral)** モード
: 一時 (Ephemeral) Diffie-Hellman は、すべての接続に対して、PFS を有効にする新規の一時 DH 秘密鍵を生成します。両方の側が新規接続に対して必ず新規 DH 秘密鍵を作成するモードは、「*Ephemeral-Ephemeral*」という用語で表現されます。

z/VSE では、一時 (Ephemeral) モードのみがサポートされます。 VSE 側は、新規接続に対して必ず新規 DH 秘密鍵を作成します。 API に対して事前生成された DH 秘密鍵を提供することはできません。関連するすべての SSL 暗号スイートは、DHE-RSA という接頭部が付けられています。次のセクションでは、z/VSE で DHE-RSA をセットアップして使用する方法を示します。

z/VSE 上での OpenSSL を用いた DHE-RSA の使用

z/VSE 5.2 以降、OpenSSL は、LE/C マルチプレクサーを使用して SSLPHASE を構成することで、z/VSE 上のすべての IP スタックと一緒に使用できるようになりました。

さらに、OpenSSL は、IPv6/VSE とそのユーティリティーである BSTTATLS および BSTTPRXY と一緒に使用できます。次のセクションでは、実際的な例を用いて、DHE-RSA ベースの SSL 暗号スイートの使用方法を示します。

DH パラメーターの生成

Diffie-Hellman をセットアップするときの最初のタスクは、*p* (大きな素数) および *g* (ジェネレーター値、OpenSSL の場合は必ず 2) という 2 つの数字で構成される、一連の DH パラメーターを生成することです。パラメーター生成は CPU を消費するため、通常は事前に一度実行されます。

これらのパラメーターを生成する場合には、次の 2 つのオプションがあります。

- ワークステーションで OpenSSL を使用する。
- Keyman/VSE ツールを使用する。

これらのパラメーターに基づいて、異なる一時セッション鍵が SSL 接続用に作成されます。

注: セキュリティー上の理由から、z/VSE 上の OpenSSL は、最小で 1024 ビット長の DH パラメーターを必要とします。

ワークステーション上での OpenSSL の使用

以下の OpenSSL コマンドは、新規 DH パラメーターが含まれる .pem ファイルを生成します。

```
openssl dhparam -out dhparam.pem 1024
```

このパラメーターは、Base64 エンコードのテキスト形式で格納され、以下の例のようになります。

```
D:¥>type dhparam.pem
-----BEGIN DH PARAMETERS-----
MIGHAoGBANclzZUHl2R0NYH5D4cIHcfM8ATuk75NeO2iaV3FhcAAfs9ljlOuJaVn
UDH9qdl9A4YrDi3VPm55r/YHA4v3wx42Xaq4YfbljeGOKfT6HuhIVS9/n3ZjwNFe
2IAJeiV4VCRAmjVrgZcUodpEK+jEH4tULNS3NO3p6BbvU/6gyCQLAgEC
-----END DH PARAMETERS-----
```

DHE-RSA ベースの SSL 暗号スイートを VSE 上で有効にするには、このテキスト形式を .pem ファイルの末尾にコピー・アンド・ペーストするだけで済みます。

Keyman/VSE ツールの使用

Keyman/VSE ツールを使用すれば、さらに簡便に DH パラメーターを作成して .pem ファイルに追加することができます。 図 42 に示すように、Keyman/VSE には DH パラメーターを生成するためのツールバー・ボタンが備えられています。

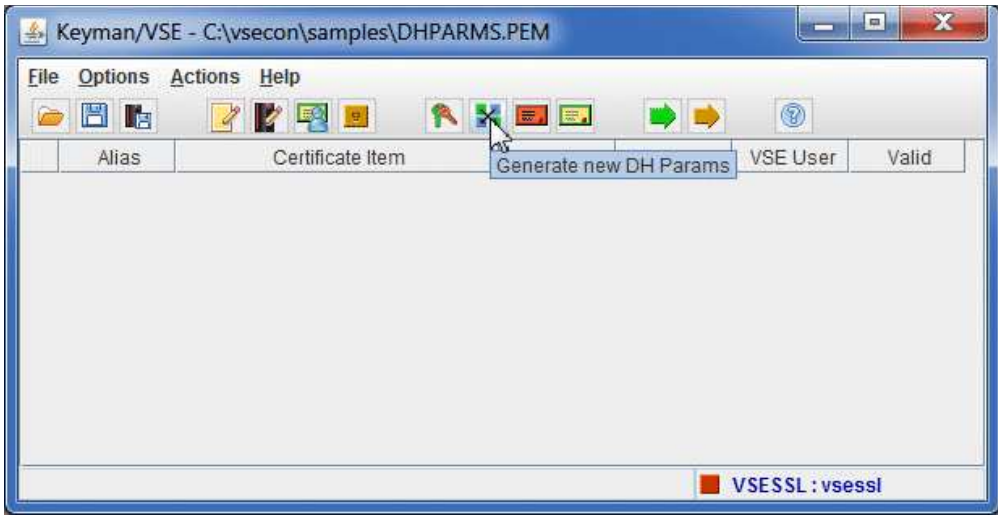


図 42. *Keyman/VSE での DH パラメーターの生成*

「**Generate new DH Params**」をクリックすると、 507 ページの図 43 のダイアログ・ボックスが表示されます。

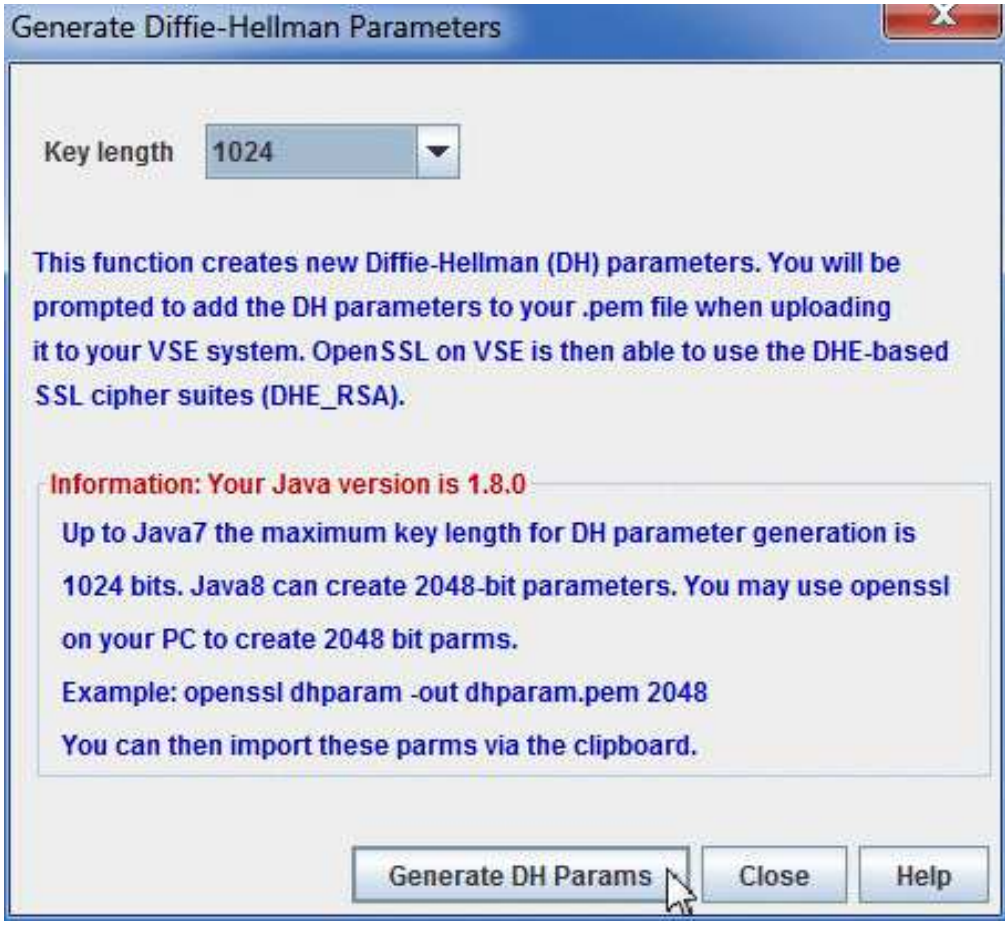


図 *43. Keyman/VSE* の「*Generate Diffie-Hellman Parameters*」ダイアログ・ボックス

Java の制限事項として、Java 8 より前のすべてのバージョンの Java では、DH パラメーター生成用の最大鍵長は 1024 ビットとなっています。それより長い鍵長を必要とする場合は、ワークステーション上で Java 8 または OpenSSL のいずれかを直接使用し、クリップボードを用いてそれらのパラメーターをインポートする必要があります。

DH パラメーターの作成またはインポートは、RSA 鍵と SSL 証明書の処理の前であっても後であっても構いません。 508 ページの図 44 は、すべての項目セットとして、RSA 鍵、CA ルート証明書、SSL サーバー証明書、および DH パラメーターを示しています。

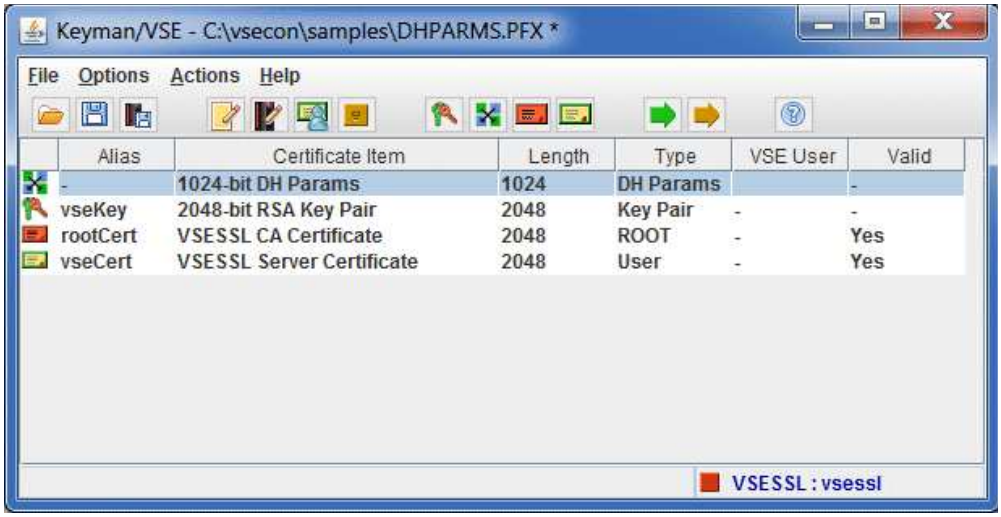


図 *44. Keyman/VSE* のすべての項目セット

`.pem` ファイルを VSE にアップロードすると、図 44 に示すとおり、DH パラメーターを `.pem` ファイルに含めることができます。これにより、DHE-RSA ベースの暗号スイートが VSE 上で使用可能になります。「**File**」⇒「**Save as PEM file on VSE**」を選択します。

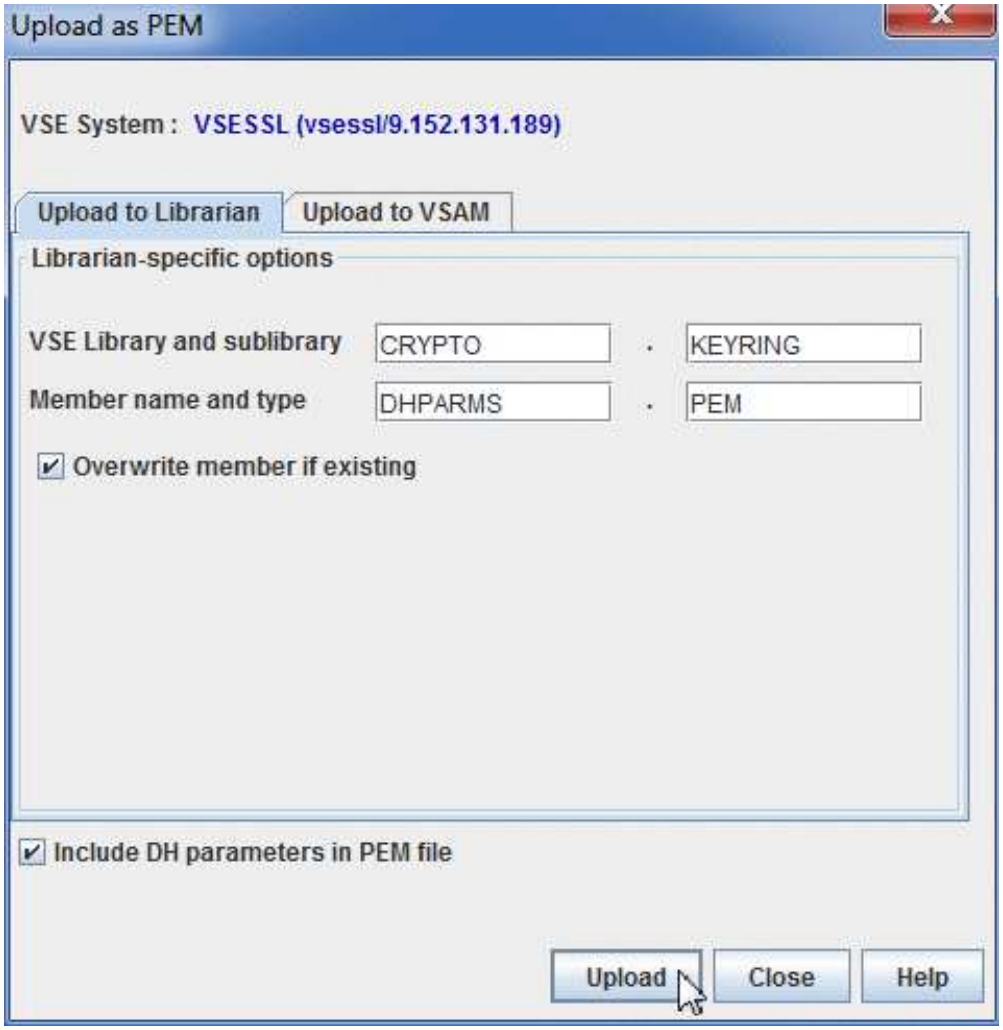


図 *45. VSE* への *PEM* ファイルのアップロード

「**Upload**」をクリックすると、すべての項目が VSE 上の新規 `.pem` ファイル内にアップロードされます。 `.pem` ファイルは、VSE/Librarian メンバーまたは VSAM ファイルのいずれかにすることができます。 VSE コネクター・サーバーが始動していることを確認します。 DH パラメーターは、 510 ページの図 46 に示すように、VSE 上の `.pem` ファイルの末尾に追加されます。

```
DITTO/ESA for VSE LE - Library Member Edit

Member DHPARMS.PEM    Library CRYPTO.KEYRING    Col 1    Format CHAR
                            SYSIPT data NO
1...5...10....5...20....5...30....5...40....5...50....5...60....5...70..
00066 9w0BAQUFAAOCAQEAlWLl5WIoiWGfS90yJaSJmcMKsUNujXSMnH2cSm8jnIuCwCKb
00067 vKBaYOeiX3QAJdW9z0NO68E7nDgEQ8IPrWz4bOj2EhffFPLZfNTwgX0iUj/AImhd
00068 cqoRVaQgmWAjj6qMY9FZWNK6RNb310umEexZVPas35wIUIOZtjrXAhpo0c9nMWrd
00069 4QVnc7Nx4JYK1Z1h8mKDc5UeSCpdIfa/+OnMXw9SbjoYTt9Hm2LpbqaQr0D3Z6Ur
00070 9FUaUSUcnBOOYLeUiO5iWofy4p2A3EOj4nuEIch+wwrf4E7GwoeniE/wCAAGiwxg
00071 dJa0lPAL2QLudmDs94L2RvgOpV36cBLEl0XmJw==
00072 -----END CERTIFICATE-----
00073 -----BEGIN DH PARAMETERS-----
00074 MIGHAoGBANclzZUHl2R0NYH5D4cIHcfM8ATuk75NeO2iaV3FhcAAfs9ljlOuJaVn
00075 UDH9qdl9A4YrDi3VPm55r/YHA4v3wx42Xaq4YfbljeGOKfT6HuhIVS9/n3ZjwNFe
00076 2IAJeiV4VCRAmjVrgZcUodpEK+jEH4tULNS3NO3p6BbvU/6gyCQLAgEC
Chapter 5. OpenSSL 183
00077 -----END DH PARAMETERS-----
00078 **** End of data ****
```

図 *46. VSE* 上の *DH* パラメーターが含まれる *PEM* ファイル

注**:** VSE がサーバーである場合、DH パラメーターは .pem ファイルにのみ含めるようにしてください。 VSE がクライアントである場合 (LDAP クライアントまたは FTP クライアントなど)、セッション・セットアップ時に DH パラメーターを提供するのはリモート・サーバーです。VSE 上の OpenSSL は、サーバーにより要求された場合に、DHE-RSA を透過的に使用します。

Java ベース・コネクターでの DHE-RSA の使用

Java はすべての DHE-RSA 関連 SSL 暗号スイートをサポートしますが、OpenSSL とは異なる名前を使用します。

暗号スイート名の中のパーツを区切るために、Java は下線文字 『_』 を使用しますが、OpenSSL はダッシュ 『-』 を使用します。表 13 は、OpenSSL および Java の、対応する暗号スイート名を示しています。

表 *13. OpenSSL* 用および *Java* 用の *DHE-RSA* 暗号スイート名

16 進コード	**OpenSSL**	**Java**
0x16	EDH-RSA-DES-CBC3-SHA	SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
0x33	DHE-RSA-AES128-SHA	TLS_DHE_RSA_WITH_AES_128_CBC_SHA
0x39	DHE-RSA-AES256-SHA	TLS_DHE_RSA_WITH_AES_256_CBC_SHA
0x67 (1)	DHE-RSA-AES128-SHA256	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
0x6B (1)	DHE-RSA-AES256-SHA256	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256

(1) 暗号スイート 67 および 6B には、TLSv1.2 が必要です。

注**:** VSE コネクター・クライアントの DHE-RSA サポートは、APAR を使用して追加されます。最新のコネクター・クライアント・バージョンであることを確認してください。

続くいくつかのセクションでは、VSE コネクター・サーバーと VSE ナビゲーターを、Java ベース・コネクター用の DHE-RSA の構成方法の例に使用します。

クライアント・サイド構成

VSE ナビゲーターの場合には、SSL プロパティー・ファイル内で以下の変更を加えます。

```
KEYRINGFILE=c:¥¥vsecon¥¥samples¥¥dhparms.pem
SSLVERSION=TLSv1.2
CIPHERSUITES=TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,TLS_DHE_RSA_WITH_AES_128_CBC_SHA2
56,TLS_DHE_RSA_WITH_AES_128_CBC_SHA
```

注: Java での AES-256 の使用には、Java 用の無制限セキュリティー強度ファイルが必要です。これらのファイルは、以下の Oracle Web サイトからダウンロードできます。

```
http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html
```

IPv6/VSE の VSE 側の構成

IPv6/VSE では、特定の SSL 暗号スイートを選択することはできません。したがって、DHE-RSA の使用は、リモート・サーバーまたはクライアントの構成に応じて異なるものになります。

- VSE がサーバーである場合は、クライアントは特定の暗号スイートを要求できます。
- VSE がクライアントである場合は、サポートされるすべての暗号スイートがネゴシエーションのためにサーバーに送信されます。

Diffie-Hellman サポートを有効にするには、BSTTATLS または BSTTPRXY に対する KEYFILE パラメーターの指定どおりに、.pem ファイル内の DH パラメーターが使用可能でなければなりません。 OpenSSL トレースは、以下のように DH サポートが使用可能かどうかを示します。

```
*** DH parameters read successfully. DHE-RSA cipher suites are available.
```

LE/C マルチプレクサーの VSE 側の構成

VSE コネクター・サーバーの SSL 構成メンバー (ICCF ライブラリー 59 の SKVCSSSL) で、SSL バージョン、.pem ファイル名、および SSL 暗号スイートを、事前に提供されているパラメーターを使用して指定します。ただし、OpenSSL を使用する場合には、以下のようにパラメーター値はさらに多くなります。

```
SSLVERSION = SSL30 | SSLV3 | TLS31 | TLSV1(1) | TLSV1.2(1) | ALL(1)
KEYRING = CRYPTO.KEYRING
CERTNAME = DHPARMS <- .pem ファイルの名前
SESSIONTIMEOUT = 86400
AUTHENTICATION = SERVER
```

(1) これらのパラメーターは、OpenSSL でのみ使用可能です。

VSE コネクター・サーバーに対して DHE-RSA を使用可能にするには、以下のようにして CIPHERSUITES パラメーターに DHE-RSA 暗号スイートを追加する必要があります。

```
CIPHERSUITES = ; COMMA SEPARATED LIST OF NUMERIC VALUES
0A, ; RSA1024_3DESCBC_SHA
2F, ; TLS_RSA_WITH_AES_128_CBC_SHA (SINCE TCP/IP 1.5E)
35, ; TLS_RSA_WITH_AES_256_CBC_SHA (SINCE TCP/IP 1.5E)
```

```
3C, ; TLS_RSA_WITH_AES_128_CBC_SHA256 (requires TLSv1.2)
39, ; TLS_DHE_RSA_WITH_AES_256_CBC_SHA (requires Diffie-Hellman)
67 ; TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (requires TLSv1.2 and DH)
```

.pem ファイル内で DH パラメーターが使用可能でなければなりません。

楕円曲線暗号化

Java はすべての DHE-RSA 関連 SSL 暗号スイートをサポートしますが、OpenSSL とは異なる名前を使用します。

楕円曲線暗号化 (ECC) は、RSA と類似の、公開鍵暗号化を提供する暗号化技法の 1 つです。 RSA のセキュリティー強度は非常に大きい素数に基づいています。それに対し、ECC は楕円曲線の数学理論を使用して、より小さな鍵を使用して同じセキュリティー・レベルを実現しています。

ECC の数学的背景については、RFC 6090 で説明されています。

`http://tools.ietf.org/html/rfc6090`

SSL/TLS での ECC の使用は、RFC 4492 で説明されています。

`http://tools.ietf.org/html/rfc4492`

実際に、ECC はパフォーマンスの速度向上のために、多くの場合 Diffie-Hellman と一緒に使用されます。 ECC は、通信パートナーの認証用の RSA に置き換わることはありませんが、EC 秘密鍵を用いた一時 (Ephemeral) DH セッション鍵の生成に使用されます。 RSA は認証を提供するために引き続き使用されます。関連する SSL 暗号スイートはすべて、名前に ECDHE-RSA が含まれており、単純な DHE ベースの暗号スイートを補完します。

単純な Diffie-Hellman (DHE-RSA) の代わりに、楕円曲線暗号化と Diffie-Hellman (ECDHE-RSA) を一緒に使用することの主要な利点は、鍵のビット数を減らしながら、より高いパフォーマンスと同じセキュリティー・レベルを実現できることです。欠点は、EC 鍵を作成して維持するという追加作業が発生することです。

次のセクションでは、z/VSE 上での ECDHE-RSA のセットアップ方法と使用方法を示します。

z/VSE 上での OpenSSL を用いた ECDHE-RSA の使用

このセクションでは、z/VSE 上で ECDHE-RSA ベースの SSL 暗号スイートを使用する方法を示します。

OpenSSL バージョン 1.0.0 以降には、以下の 2 つの異なる ECC 実装があります。

- z/VSE 上で使用される 32 ビットの実装。
- 64 ビットの実装。これはパフォーマンスは向上しますが、最新の 64 ビット gcc コンパイラーを必要とし、z/VSE 上では使用できません。z/VSE 上の OpenSSL コードは、`OPENSSL_NO_EC_NISTP_64_GCC_128` を使用してコンパイルされます。

次のセクションでは、EC 鍵を作成して、OpenSSL で使用するために z/VSE にアップロードする方法を示します。

EC 鍵の生成

このセクションでは、z/VSE 上で ECDHE-RSA ベースの SSL 暗号スイートを使用する方法を示します。

EC 鍵の生成は、ワークステーション上の OpenSSL を使用して行うことができますが、Keyman/VSE ユーティリティーを使用して行うこともできます。

EC 鍵を作成する OpenSSL コマンドの例を以下に示します。

```
openssl ecparam -out ecparam.pem -name prime256v1
openssl genpkey -paramfile ecparam.pem -out ecdhkey.pem
```

Keyman/VSE ユーティリティーを使用する場合は、Keyman メインウィンドウの「Create new EC key」ボタンを押します。

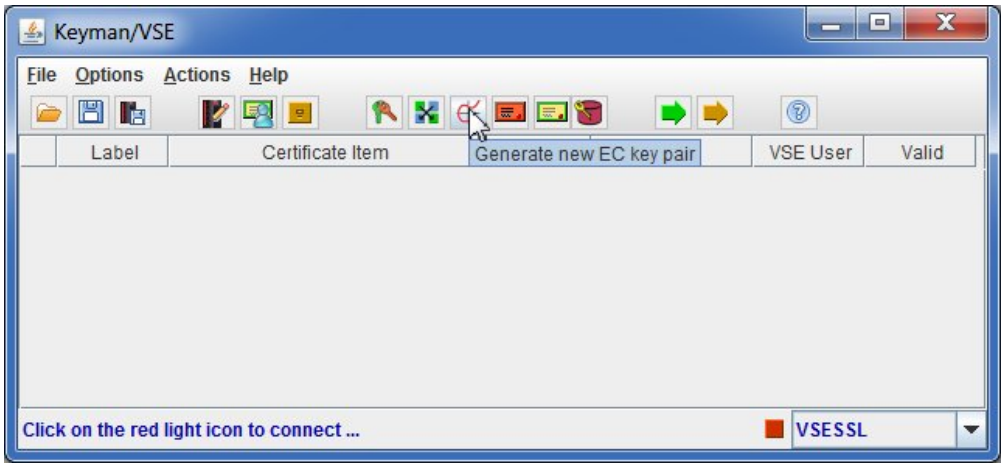


図 *47. Keyman* メインウィンドウ

ドロップダウン・リスト・ボックスから「Elliptic Curve」を選択し、「Generate key」を押します。

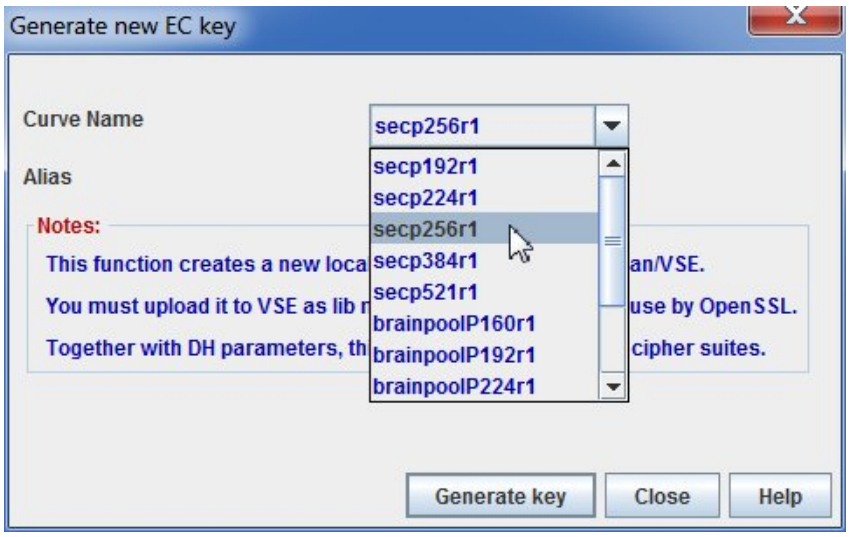


図 *48. Keyman* の「*Generate New EC Key*」ウィンドウ *(1/2)*

曲線名は EC 鍵のビット長を示します。 EC 鍵は、鍵リング・ライブラリー内のライブラリー・メンバー ECDHKEY.PEM として z/VSE にアップロードする必要があります。

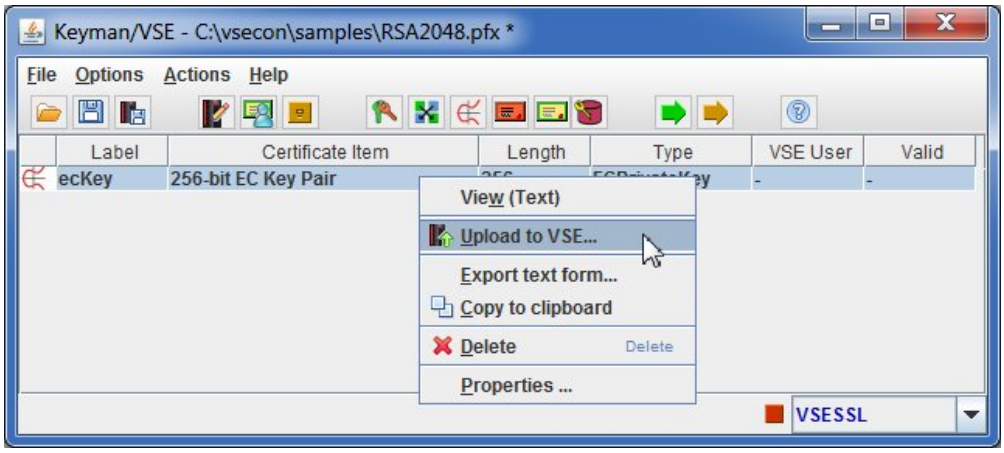


図 *49. Keyman* の「*Generate New EC Key*」ウィンドウ *(2/2)*

現在、EC 鍵は ecdhkey.pem ファイルに含まれており、内容は次のように表示されます。

```
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQg0PvMUyVFylSlfwx8
H0q1Er8ve9pgcvs6Ezfty8yq6qKhRANCAAS9d/zL/ZIwydd5EvLoLF3+GnUHQ/pu
PiN945ucTiLTj08YjZ7SCIWbGskb+DH32viG6+4goAoZQT3Tzoi3EYz8
-----END PRIVATE KEY-----
```

EC 鍵には RSA 秘密鍵と同じストリング区切り文字があるので、RSA 鍵と一緒に同じ PEM ファイルに保管することはできません。

楕円曲線の使用

以下に示すのは、Crypto Express CCA コプロセッサーまたはその後続製品と、CCA 4.2 ファームウェア・ロードまたはそれ以降を使用した場合の、ハードウェア・アクセラレーションによる楕円曲線です。

Brainpool 曲線は、OpenSSL 1.0.2 以降でサポートされています。

表 *14.* ハードウェア・アクセラレーションの楕円曲線

	p 鍵長のサイズ	
曲線	**(**ビット**)**	**(**バイト**)**
secp192r1	192	24
secp224r1	224	28
secp256r1	256	32
secp384r1	384	48
secp521r1	521	65
brainpoolP160r1	160	20
brainpoolP192r1	192	24
brainpoolP224r1	224	28
brainpoolP256r1	256	32
brainpoolP320r1	320	40
brainpoolP384r1	384	48
brainpoolP512r1	512	64

z/VSE への EC 鍵のアップロード

Keyman/VSE ユーティリティーを使用する以外にも、FTP や、ASCII から EBCDIC への変換機能を備えたその他のファイル転送メカニズムを使用して、EC 鍵を含む PEM ファイルを z/VSE にアップロードできます。

また、端末エミュレーターを使用して、新規 DITTO 作成ファイルにコピー・アンド・ペーストすることもできます。 z/VSE 上のターゲット・ファイルは、VSE/Librarian メンバーでなければなりません。 z/VSE 上のメンバーの内容は、ワークステーション上の文字表現と同一の表示内容でなければなりません。以下に、DITTO 内のアップロードされたメンバーを示します。

```
DITTO/ESA for VSE          LE - Library Member Edit

Member ECDHKEY.PEM       Library CRYPTO.KEYRING     Col 1          Format CHAR
                                                                SYSIPT data NO
      1...5...10....5...20....5...30....5...40....5...50....5...60....5...70..
00000 ****  Top of data  ****
00001 -----BEGIN PRIVATE KEY-----
00002 MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQg0PvMUyVFylSlfwx8
00003 H0q1Er8ve9pgcvs6Ezfty8yq6qKhRANCAAS9d/zL/ZIwydd5EvLoLF3+GnUHQ/pu
00004 PiN945ucTiLTj08YjZ7SCIWbGskb+DH32viG6+4goAoZQT3Tzoi3EYz8
00005 -----END PRIVATE KEY-----
00006 ****  End of data  ****
```

次のセクションでは、Java ベース・コネクターを用いて ECDHE-RSA を使用する方法を示します。

Java ベース・コネクターでの ECDHE-RSA の使用

Java はすべての ECDHE-RSA 関連 SSL 暗号スイートをサポートしますが、OpenSSL とは異なる名前を使用します。

表 15 は、z/VSE 上でサポートされる暗号スイートと、Java 用の対応する名前を示しています。

表 *15. OpenSSL* 用および *Java* 用の *ECDHE-RSA* 暗号スイート名：

16 進コード	**OpenSSL**	**Java**
0xC012	ECDHE-RSA-DES-CBC3-SHA	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
0xC013	ECDHE-RSA-AES128-SHA	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
0xC014	ECDHE-RSA-AES256-SHA	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
0xC027 (1)	ECDHE-RSA-AES128-SHA256	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

(1) 暗号スイート C027 には、TLSv1.2 が必要です。

注**:** SHA-384 を使用する ECDHE-RSA 暗号スイートは、z/VSE 上の OpenSSL における SHA-384 および SHA-512 の一般的な制約事項を理由として、現在のところ z/VSE 上ではサポートされていません。

クライアント・サイド構成

VSE ナビゲーターの場合には、SSL プロパティー・ファイル内で以下の変更を加えます。

```
KEYRINGFILE=c:¥¥vsecon¥¥samples¥¥ecdh.pem
SSLVERSION=TLSv1.2
CIPHERSUITES=TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_256_CBC_
SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
```

RSA 鍵および DH パラメーターが含まれている PEM ファイルをここに指定します。 EC 鍵は、クライアント側では必要ありません。この鍵は、VSE 上の 2 次ファイル ECDHKEY.PEM 内にのみ置きます。

IPv6/VSE の VSE 側の構成

IPv6/VSE では、特定の SSL 暗号スイートを選択することはできません。その代わり、**gsk_get_cipher_info** から返されるストリングが、**gsk_secure_soc_init** API 関数を呼び出すときに OpenSSL に戻されます。このストリングには、サポートされるすべての ECDHE-RSA ベースの暗号スイートが含まれます。

楕円曲線サポートを使用可能にするには、BSTTATLS または BSTTPRXY に対する **KEYRING** パラメーターで指定されたのと同じ VSE サブライブラリー内の VSE ライブラリー・メンバー ECDHKEY.PEM で、EC 鍵が使用可能でなければなりません。 OpenSSL トレースは、EC 鍵が入手可能であり使用可能であるかを示します。

```
*** EC Private Key read successfully.
*** EC key set. ECDHE-RSA cipher suites are available.
```

LE/C マルチプレクサーの VSE 側の構成

一般に、TCP/IP for z/VSE および Linux Fast Path (LFP) では、OpenSSL は LE/C マルチプレクサーを用いる場合のみ使用できます。 ECDHE-RSA の可用性は、関連するアプリケーションによって異なります。例えば、VSE コネクター・サーバーを使用する場合は、以下のようにサーバーの SSL 構成メンバー (SKVCSSSL) に ECDHE 暗号スイートを指定します。

```
SSLVERSION = TLSV1.2
KEYRING = CRYPTO.KEYRING
CERTNAME = ECDHSSL
SESSIONTIMEOUT = 86400
AUTHENTICATION = SERVER
```

EC 鍵は、**KEYRING** パラメーターによって指定された VSE サブライブラリー内の 2 次メンバー ECDHKEY.PEM に含まれている必要があります。 **CIPHERSUITES** パラメーターには、以下の ECDHE 関連暗号スイートが含まれています。

```
CIPHERSUITES = ; COMMA SEPARATED LIST OF NUMERIC VALUES
2F, ; TLS_RSA_WITH_AES_128_CBC_SHA
35, ; TLS_RSA_WITH_AES_256_CBC_SHA
67, ; TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
6B, ; TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
C013, ; TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
C014, ; TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
C027, ; TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
```

制約事項

このセクションでは、z/VSE 上で OpenSSL を使用した場合の制約事項および既知の問題を説明します。

SHA-512 サポートがない

VSE C コンパイラーは、SHA-512 関連モジュールをコンパイルできません。 OpenSSL FAQ.txt ファイルには、次のように記述されています。

「OpenSSL SHA-512 の実装は、64 ビット整数型のコンパイラー・サポートの場合は異なります。旧式のコンパイラー [ULTRIX cc、SCOcompiler など] はこのサポートがないため、当該のモジュールをコンパイルできません。 no-sha512 を ./config [または ./Configure] コマンド・ラインに追加して、SHA-512 を無効にすることをお勧めします。別の方法として、GCC に切り替えることもできます。」

現時点では、VSE コードは OPENSSL_NO_SHA512 オプションを使用してコンパイルされます。これは、CPACF によって提供される SHA-512 を使用して変更できます。

OpenSSL 速度テストの実行

OpenSSL では、RSA、AES、SHA、DES など、暗号アルゴリズムを実行したときのシステムの速度をチェックできる組み込みの速度テストが提供されます。

z/VSE では、このテストはフェーズ IJBSSL に含まれており、次の JCL を使用して呼び出すことができます。

```
// EXEC SPEEDTST,PARM='OPENSSL'
```

サポートされるパラメーターのリストを表示するには、OpenSSL がインストールされているワークステーションで次のコマンドを入力します。

```
#openssl speed ?
```

コマンドの詳細については、http://www.openssl.org/ を参照してください。

テストの呼び出し

z/VSE で RSA テストを呼び出すには、次のどちらかを指定します。

```
// EXEC SPEEDTST,PARM='OPENSSL RSA'
```

または

```
// EXEC SPEEDTST,PARM='RSA'
```

他のアルゴリズムを呼び出す例を以下に示します。

```
// EXEC SPEEDTST,PARM='AES-128-CBC'
// EXEC SPEEDTST,PARM='SHA1'
// EXEC SPEEDTST,PARM='DES-EDE3 SHA256 AES-128-CBC'
```

以下のアルゴリズムが暗号ハードウェアによってサポートされます。

- RSA (RSA-1024、RSA-2048、RSA-4096 を含む)
- DES-CBC
- DES-EDE3
- AES-128-CBC
- AES-192-CBC
- AES-256-CBC
- SHA1
- SHA256
- ECDHE-RSA

z/OS SSL API

z/OS SSL API は、$EDCTCPV フェーズを介した TCP/IP for z/VSE、およびフェーズ IJBSSL を介した OpenSSL でサポートされます。

この API の詳細については、以下を参照してください。

- 102 ページの『TCP/IP 呼び出し可能関数 — 関数の説明』
- z/OS Cryptographic Services System SSL プログラミング

サポートされる API 関数呼び出しと、GSK レイヤーへの OpenSSL の実装により生じる相違点を以下に示します。

gsk_free_memory

SSL ランタイムによって割り当てられていたストレージを解放します。

z/OS と比較して、z/VSE に関する変更点はありません。

gsk_get_cipher_info

サポートされる暗号仕様を返します。

z/VSE に関する変更点は以下のとおりです。

- 返される暗号スイートのリストが z/OS マニュアルに記載されたものと異なります。z/OS で使用される一部の暗号が OpenSSL でサポートされないためです (例: "00")。z/VSE 上の OpenSSL は以下のストリングを返します。

```
"0A090815120306"  // LOW_SECURITY
"C027C014C013C0126B673933163D3C352F"  // HIGH_SECURITY
```

- **gsk_sec_level** 構造体のバージョン・フィールドは、現在サポートされている OpenSSL バージョンを返します。例えば、101 = 1.0.1 などです。

示されている暗号スイートの一部は、特定の OpenSSL バージョンでサポートされていない場合があります。

gsk_get_dn_by_label
証明書の識別名を取得します。

z/VSE に関する変更点は以下のとおりです。

- 指定された鍵/証明書ファイルは、メンバータイプ PEM のライブラリアン・メンバーか、VSAM ファイルである必要があります。
- z/OS では、キー・データベースにアクセスできない場合、NULL が返されます。ただし、z/VSE では、鍵ストアにアクセスするための十分な情報はここにありません。

gsk_initialize
システム SSL のランタイム環境を初期化します。

z/VSE に関する変更点は以下のとおりです。

- JCL 変数 SSL$CPH、SSL$CSI、SSL$DBG、SSL$ICA、および SSL$TRC を読み取って評価します。

gsk_secure_soc_close
セキュア・ソケット接続を閉じます。

z/VSE に関する変更点はありません。

gsk_secure_soc_init
セキュア・ソケット接続を初期化します。

z/VSE に関する変更点はありません。

gsk_secure_soc_read
セキュア・ソケット接続を使用して、データを読み取ります。

z/VSE に関する変更点は以下のとおりです。

- 呼び出し元で `buflen = 0` を指定し、保留バイトがあるかどうかを確認できます。 `buflen = 0` の場合、SSL_pending が呼び出され、**gsk_secure_soc_read** は `SSL_pending` で返された戻りコードを返します。

gsk_secure_soc_reset
セキュア接続のセッション鍵をリセットします。

z/VSE に関する変更点はありません。

gsk_secure_soc_write
セキュア・ソケット接続を使用して、データを書き込みます。

z/VSE に関する変更点はありません。

gsk_uninitialize

SSL 環境を終了します。

z/VSE に関する変更点はありません。

gsk_user_set

アプリケーション・コールバックを設定します。

現在、z/VSE ではサポートされていません。

第 5 部 CICS リスナー・サポート

第 18 章 CICS リスナー・サポートのセットアップと構成

概説

このセクションでは、CICS リスナー・サポートを構成するのに必要な手順を説明します。エラー・メッセージは、「*IBM z/VSE メッセージおよびコード 第 1 巻*」に記載されています。

重要:

1. CICS リスナー・サポートでは SIT パラメーター SVA=YES を指定して CICS/TS を開始する必要があります。
2. IBM 提供の CICS リスナー・サポートは、「タスク関連ユーザー出口」プログラム EZATRUE を必要とします。このプログラムの開始方法について詳しくは、 93 ページの『EZA インターフェースに関する CICS の考慮事項』を参照してください。
3. デュアル・スタック・サポートがアクティブである場合、IPv6 TCP/IP スタックで使用できるのは、IBM 提供の CICS リスナー・プログラム **EZACIC02** のみです。詳しくは、 45 ページの『第 8 章 IPv6/VSE 概要』を参照してください。デュアル・スタック・サポートがアクティブでない場合には、予測できない結果が起こる可能性があります。例えば、BIND 要求が ERRNO 10218 で拒否され、メッセージ EZY1369E が発行されるなどです。

CICS リスナー・サポートを開始するには、その前に以下を実行する必要があります。

タスク	参照先
RDO を使用して、追加のファイル、プログラム、マップ、および一時データを CICS に定義する。	『CICS — CICS リソースの定義』
構成マクロ (EZACICD) を使用して、CICS リスナー構成データ・セットを構築する。	529 ページの『構成マクロ (EZACICD) を使用した構成データ・セットの構築』
構成トランザクションを使用して、構成データ・セットをカスタマイズする。	535 ページの『構成データ・セットのカスタマイズ』
注: CICS 稼働中に EZAC を使用してデータ・セットを変更できます。 535 ページの『構成トランザクション (EZAC)』を参照してください。	

CICS — CICS リソースの定義

CICS リスナー・サポートには、以下の定義が必要です。

- 524 ページの『トランザクション定義』
- 525 ページの『プログラム定義』
- 526 ページの『ファイル定義』
- 526 ページの『一時データ定義』

注: z/VSE では、これらすべての定義は、IJSYSRS.SYSLIB 内のメンバー IESCSEZA.Z と IESZDCT.A を使用してアクティブ化されています。このセットアップには、トランザクションに対する TASKDATAKEY(CICS) の定義と、プログラムに対する EXECKEY(CICS) の定義が含まれます。これは CICS ストレージ保護で稼働する時に必要です。これらの定義は、CICS ストレージ保護なしで稼働するときには無視されます。

トランザクション、プログラム、およびファイルを CICS オンライン・リソース定義 (Resource Definition Online (RDO)) ファシリティーに対して定義することについて詳しくは、「CICS TS for VSE/ESA, Resource Definition Guide」を参照してください。

トランザクション定義

CICS リスナーをサポートするためには、以下の 4 つのトランザクションが必要です。

EZAC ソケット・インターフェースを構成する

EZAO
ソケット・インターフェースを使用可能にする

EZAP ソケット・インターフェースの終了中に呼び出される内部トランザクション

EZAL
リスナー・タスク

注: これは、単一のリスナーです。同じ CICS パーティション内の各リスナーは、それぞれ固有のトランザクション ID を必要とします。

ヒント: トランザクション EZAL、EZAO、および EZAP では、優先順位 255 が定義されています。これによって、トランザクションのディスパッチがタイムリーに行われることが保証され、EZAL の場合には、クライアント要求サービスの接続率が最大化されます。

ストレージ保護の使用

CICS ストレージ保護を設定して稼働している場合、EZAP、EZAO、および EZAL トランザクションは、TASKDATAKEY(CICS) で定義されなければなりません。これが行われていない場合、EZAO は ASRA 異常終了コードで失敗します。このコードは、EZACIC01 が CDSA を不正に上書きしようとしたことを示します。

マシンがストレージ保護をサポートしていない場合、または、マシンがストレージ保護を使用できない場合、TASKDATAKEY(CICS) は無視され、エラーは発生しません。

注:

1. IBM 提供リスナーの使用は必要ありません。
2. EZAL 以外のトランザクション名を使用してもかまいません。
3. EZAO と EZAP に対する TASKDATALoc 値と、EZAL に対する TASKDATALoc 値は、同じでなければなりません。

プログラム定義

以下のプログラムおよび 1 つのマップ・セットが必要です。

EZACIC00
接続マネージャー・プログラムです。これは、トランザクション EZAO と EZAP を介して CICS TCP/IP を使用可能または使用不可にします。

EZACIC01
タスク関連ユーザー出口 (TRUE) です。

EZACIC02
トランザクション EZAL によって使用されるリスナー・プログラムです。このトランザクションは、EZAO トランザクションを介して CICS TCP/IP リスナーを使用可能にしたときに開始します。

注**:** IBM 提供リスナーを使用する必要がなくても、リスナー機能を用意する必要はあります。

EZACIC20
CICS リスナー・インターフェース用の初期設定/終了フロントエンド・モジュールです。

EZACIC21
CICS リスナー・インターフェース用の初期設定モジュールです。

EZACIC22
CICS リスナー・インターフェース用の終了モジュールです。

EZACIC23
構成トランザクション (EZAC) 用の主モジュールです。

EZACIC24
トランザクション EZAC と EZAO 用のメッセージ配信モジュールです。

EZACIC25
ドメイン・ネーム・サーバー (DNS) キャッシュ・モジュールです。

EZACICME
米国英語テキスト配信モジュールです。

EZACICM
トランザクションによって使用されるすべてのマップが含まれています。

EZASOH00、EZASOH99
CICS リスナーによって使用される TCP/IP API 用のインターフェース・モジュールです。

ストレージ保護の使用

CICS ストレージ保護を設定して稼働している場合、必要なすべての CICS リスナー・プログラムの CEDA 定義の一部として、EXECKEY=CICS が含まれていなければなりません。

マシンがストレージ保護をサポートしていない場合、または、マシンがストレージ保護を使用可能でない場合、EXECKEY(CICS) は無視され、エラーは発生しません。

ファイル定義

CICS に対するアップデートには、2 つのファイルがあります。1 つは、CICS リスナー構成ファイル EZACONF であり、もう 1 つは、ドメイン・ネーム・サーバー・キャッシュ機能 (EZACIC25) を使用する場合に必要な EZACACH です。

一時データ定義

CICS リスナー・サポートは、メッセージ用に一時データ待ち行列を使用します。z/VSE では、EZAM 一時データ待ち行列が事前定義済みであり、CICS リスナー・サポートでもユーザー自身のソケット・アプリケーションでもこれを使用することができます。一時データ待ち行列の名前は、変更することができます。

変更する場合、**EZAC DEFINE CICS** ダイアログ、**EZACICD TYPE=CICS** マクロ呼び出し、あるいはその両方の **ERRORTD** パラメーターに指定されている名前と一致しなければなりません。 529 ページの『構成マクロ (EZACICD) を使用した構成データ・セットの構築』を参照してください。

592 ページの『リスナーの入力フォーマット (標準リスナーのみ)』に説明があるように、リスナー・トランザクションは一時データ待ち行列を使用してサーバーを開始できます。以下に、DCT のトリガー・レベルの仕組みを使用して開始される、アプリケーションに対する DCT 項目を示します。

```
DFHDCT TYPE=INTRA,                                                     X
      DESTID=TRAA,                                                     X
      DESTFAC=FILE,                                                    X
      TRIGLEV=1,                                                       X
      TRANSID=TRAA
            ...
            ...
```

図 *50. CICS TCP/IP* が必要とする、*DCT* への追加

CICS のモニター

オプションで、CICS リスナー・インターフェースは、CICS モニター機能を使用して、操作に関するデータを収集します。リスナーは、ID が「EZA02」のイベント・モニター・ポイント (Event Monitoring Point (EMP)) を使用して、パフォーマンス・クラス・データを収集します。

リスナー用のイベント・モニター・ポイント

リスナーは、接続の受け入れおよびサーバー・タスク・スタートアップに関連するアクティビティーをモニターします。

リスナーは、以下のイベントをカウントします。

- 受け入れた接続要求の数
- 開始したトランザクションの数
- トランザクション ID が無効であるためにリジェクトされたトランザクションの数

- トランザクションが使用不可であるためにリジェクトされたトランザクションの数
- プログラムが使用不可であるためにリジェクトされたトランザクションの数
- givesocket の失敗のためにリジェクトされたトランザクションの数
- セキュリティー出口からの否定応答のためにリジェクトされたトランザクションの数
- 実行が認可されていないトランザクションの数
- 入出力エラーのためにリジェクトされたトランザクションの数
- スペースがないためにリジェクトされたトランザクションの数
- TD 長さエラーのためにリジェクトされたトランザクションの数

以下のモニター管理テーブル (Monitor Control Table (MCT)) 項目は、リスナーによって使用されるパフォーマンス・クラス内のイベント・モニター・ポイントを利用します。

図 *51.* リスナー用のモニター管理テーブル *(MCT)*

```
DFHMCT TYPE=EMP,ID=(EZA02.01),CLASS=PERFORM,                          X
       PERFORM=ADDCNT(1,1)
DFHMCT TYPE=EMP,ID=(EZA02.02),CLASS=PERFORM,                          X
       PERFORM=ADDCNT(2,1)
DFHMCT TYPE=EMP,ID=(EZA02.03),CLASS=PERFORM,                          X
       PERFORM=ADDCNT(3,1)
DFHMCT TYPE=EMP,ID=(EZA02.04),CLASS=PERFORM,                          X
       PERFORM=ADDCNT(4,1)
DFHMCT TYPE=EMP,ID=(EZA02.05),CLASS=PERFORM,                          X
       PERFORM=ADDCNT(5,1)
DFHMCT TYPE=EMP,ID=(EZA02.06),CLASS=PERFORM,                          X
       PERFORM=ADDCNT(6,1)
DFHMCT TYPE=EMP,ID=(EZA02.07),CLASS=PERFORM,                          X
       PERFORM=ADDCNT(7,1)
DFHMCT TYPE=EMP,ID=(EZA02.08),CLASS=PERFORM,                          X
       PERFORM=ADDCNT(8,1)
DFHMCT TYPE=EMP,ID=(EZA02.09),CLASS=PERFORM,                          X
       PERFORM=ADDCNT(9,1)
DFHMCT TYPE=EMP,ID=(EZA02.10),CLASS=PERFORM,                          X
       PERFORM=ADDCNT(10,1)
DFHMCT TYPE=EMP,ID=(EZA02.11),CLASS=PERFORM,                          X
       PERFORM=ADDCNT(11,1)
DFHMCT TYPE=EMP,ID=(EZA02.12),CLASS=PERFORM,                          X
       PERFORM=(MLTCNT(1,11)),                                        X
       COUNT=(1,CONN,STARTED,INVALID,DISTRAN,DISPROG,GIVESOKT,SECEXIT)
```

ID パラメーターでは次の仕様が使用されます。

(EZA02.01)
ACCEPT 呼び出しの完了。

(EZA02.02)
CICS トランザクション開始の完了。

(EZA02.03)
無効なトランザクション ID の検出。

(EZA02.04)
使用不可にされたトランザクションの検出。

(EZA02.05)
使用不可にされたプログラムの検出。

(EZA02.06)
givesocket 失敗の検出。

(EZA02.07)
セキュリティー出口によるトランザクションの拒否。

(EZA02.08)
トランザクションが認可されていない

(EZA02.09)
トランザクション開始での入出力エラー。

(EZA02.10)
TD 開始メッセージ用のスペースなし

(EZA02.11)
TD 長さエラー

(EZA02.12)
プログラム終了。

CICS プログラム・リスト・テーブル (PLT)

PLT に更新を加えることによって、CICS リスナー・インターフェースの自動的なスタートアップ/シャットダウンを行うことができます。これを行うには、適切な PLT 内に EZACIC20 モジュールを置きます。

CICS リスナー・インターフェースを自動的に開始するには、PLTPI 内の以下の項目を DFHDELIM 項目の後に 置きます。

```
DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
```

CICS リスナー・インターフェースを自動的にシャットダウンするには、PLTSD 内の以下の項目を DFHDELIM 項目の前に 置きます。

```
DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
```

CICS TCP/IP 環境の構成

CICS リスナー構成ファイル (EZACONF) には、CICS リスナー環境に関する情報が入っています。

このファイルは、CICS インスタンスと、それらのインスタンス内のリスナーという、2 つのタイプのオブジェクトから編成されています。このデータ・セットの作成は、次の 3 段階で行われます。

1. VSAM IDCAMS (アクセス方式サービス・プログラム) を使用して、空のデータ・セットを作成します。z/VSE の場合、構成ファイルは事前割り振り済みですが、空です。

   これらは、事前割り振りの VSAM 定義ステートメントです。

   ```
   DEFINE CLUSTER(NAME(VSE.EZACICS.CONFIG)   -
        RECORDS(3000 2000)                    -
        SHAREOPTIONS(2)                       -
   ```

```
    RECORDSIZE(150,150)                 -
    VOLUMES(SYSWK1,DOSRES)              -
    NOREUSE                             -
    INDEXED                             -
    FREESPACE(15 7)                     -
    KEYS (16,0)                         -
    NOCOMPRESSED                        -
    TO (99366))                         -
    DATA (NAME(VSE.EZACICS.CONFIG.@D@)  -
    CONTROLINTERVALSIZE (4096))         -
    INDEX (NAME(VSE.EZACICS.CONFIG.@I@))  -
    CATALOG (VSESP.USER.CATALOG)
```

2. そのデータ・セットを EZACICD マクロによって生成されたプログラムを使用して初期設定します。ライブラリー 59 内のメンバー SKCICSLI にある、構成ファイルを初期設定するサンプル・ジョブを参照してください。
3. 構成トランザクション EZAC を使用して、データ・セットに追加または変更します。このステップについては、 535 ページの『構成データ・セットのカスタマイズ』に説明があります。[1]

構成マクロ (EZACICD) を使用した構成データ・セットの構築

構成データ・セットを構築するため、構成マクロ (EZACICD) が使用されます。

その後、このデータ・セットは、RDO を使用して CICS に取り込むこと、そして構成トランザクションを使用して変更することができます。 535 ページの『構成トランザクション (EZAC)』を参照してください。このマクロはキーワード駆動です。TYPE キーワードは、特定の機能要求を制御します。データ・セット中には、サポートされる各 CICS インスタンスごとに 1 つのレコードがあり、各リスナーごとに 1 つのレコードがあります。次に示すのは、CICS リスナー・インターフェースの 2 つのインスタンスに対して、構成ファイルを作成する際に必要なマクロの例です。

```
EZACICD TYPE=INITIAL,      Start of macro assembly input             X
      FILNAME=EAZCONF,     Name for configuration file               X
      PRGNAME=EZACONFP     Name of batch program to run
EZACICD TYPE=CICS,         CICS record definition                    X
      APPLID=DBDCCICS,     APPLID of CICS                            X
      TCPADDR=SOCKET00,    Name of TCP/IP Address                    X
      PLTSDI=YES,          PLT shutdown method is immediately        X
      CACHMIN=15,          Minimum refresh time for cache            X
      CACHMAX=30,          Maximum refresh time for cache            X
      CACHRES=10,          Maximum number of resident resolvers      X
      ERRORTD=CSMT,        Transient data queue for error msgs       X
      SMSGSUP=NO           STARTED Messages Suppressed?
EZACICD TYPE=LISTENER,     Listener record definition                X
      FORMAT=STANDARD,     Standard Listener                         X
      APPLID=DBDCCICS,     Applid of CICS region                     X
      TRANID=EZAL,         Transaction name for Listener             X
      PORT=3010,           Port number for Listener                  X
      AF=INET,             Listener Address Family                   X
      IMMED=YES,           Listener starts up at initialization?     X
      BACKLOG=20,          Backlog value for Listener                X
      NUMSOCK=50,          # of sockets supported by Listener        X
      MINMSGL=4,           Minimum input message length              X
      ACCTIME=30,          Timeout value for Accept                  X
      GIVTIME=30,          Timeout value for Givesocket              X
```

1. EZAC トランザクションは、CICS オンライン・リソース定義 (RDO) によって使用される CEDA トランザクションをモデルにしています。

```
      REATIME=30,          Timeout value for Read                    X
      RTYTIME=10,          Wait 10 seconds for TCP to come back      X
      TRANTRN=YES,         Is TRANUSR=YES conditional?               X
      TRANUSR=YES,         Translate user data?                      X
      SECEXIT=EZACICSE     Name of security exit program
EZACICD TYPE=LISTENER,     Listener record definition                X
      FORMAT=ENHANCED,     Enhanced Listener                         X
      APPLID=DBDCCICS,     Applid of CICS region                     X
      TRANID=CSKM,         Transaction name for Listener             X
      PORT=3011,           Port number for Listener                  X
      AF=INET,             Listener Address Family                   X
      IMMED=YES,           Listener starts up at initialization?     X
      BACKLOG=20,          Backlog value for Listener                X
      NUMSOCK=50,          # of sockets supported by Listener        X
      ACCTIME=30,          Timeout value for Accept                  X
      GIVTIME=30,          Timeout value for Givesocket              X
      REATIME=30,          Timeout value for Read                    X
      RTYTIME=20,          Wait 20 seconds for TCP to come back      X
      CSTRAN=TRN1,         Name of child IPv4 server transaction     X
      CSSTTYP=KC,          Child server startup type                 X
      CSDELAY=000000,      Child server delay interval               X
      MSGLEN=0,            Length of input message                   X
      MSGFORM=ASCII,       Output message format                     X
      SECEXIT=EZACICSE     Name of security exit program
EZACICD TYPE=FINAL         End of assembly input
```

TYPE パラメーター

TYPE パラメーターは、機能要求を制御します。以下の値を指定できます。

INITIAL
生成環境を初期化します。生成ごとに一度、マクロの最初の呼び出し時にこの値を定義します。**TYPE=INITIAL** が指定される場合、以下のパラメーターが適用されます。

PRGNAME=EZACONFP|*xxxxxxx*
生成される初期設定プログラムの名前。デフォルトは **EZACONFP** です。

FILNAME=EZACONF|*xxxxxxx*
初期設定プログラムの実行時に構成ファイルに使用されるファイル名。デフォルトは **EZACONF** です。

CICS
CICS オブジェクトを識別します。このオブジェクトは、1 つの特定の CICS インスタンスに対応し、構成レコードを作成します。TYPE=CICS が指定される場合、以下のパラメーターが適用されます。

APPLID=*xxx*
CICS リスナーのこのインスタンスが実行する場所になる CICS アドレス・スペースのアプリケーション ID。このパラメーターは必須です。

CACHMAX=30|*nnn*
ドメイン・ネーム・サーバー・キャッシュの最大リフレッシュ時間 (分)。この値は、ネットワークの安定度、つまり、あるドメイン名の IP アドレスが同じまま保たれると想定される時間に基づきます。この値を大きくすると、パフォーマンスは向上しますが、名前の解決時に正しくない (有効期限が切れた) アドレスを取得するリスクが増えます。この値は **CACHMIN** より大きくなければなりません。デフォルト値は 30 です。指定可能な値は 0 から 999 までです。

CACHMIN=15|*nnn*
ドメイン・ネーム・サーバー・キャッシュの最小リフレッシュ時間 (分)。この値は、ネットワークの安定度、つまり、あるドメイン名の IP アドレスが同じまま保たれると想定される時間に基づきます。この値を大きくすると、パフォーマンスは向上しますが、名前の解決時に正しくない (有効期限が切れた) アドレスを取得するリスクが増えます。この値は **CACHMAX** より小さくなければなりません。デフォルト値は 15 です。指定可能な値は 0 から 998 までです。

CACHRES=10|*nn*
必要な同時リゾルバーの最大数。同時リゾルバーの数がこの値以上になると、レコードの経過時間が **CACHMAX** の値より大きくならない限り、キャッシュ・レコードのリフレッシュは行われません。デフォルトは 10 です。指定可能な値は 0 から 99 までです。

ERRORTD=EZAM|*xxx*
エラー・メッセージの書き込み先になる一時データ宛先の名前。デフォルトは **EZAM** です。

PLTSDI=NO|YES
EZACIC20 PLT プログラムを使用して CICS TCP/IP リスナー・サポートをシャットダウンする場合、**PLTSDI** パラメーターはインターフェースをただちにシャットダウンするかどうかを指定します。デフォルトは **NO** で、据え置きシャットダウンを指定します。**YES** は、即時シャットダウンを指定します。PLTSDI パラメーターが指定されない場合は据え置きシャットダウンが実行されます。据え置きシャットダウンでは、すべての CICS リスナーが安全に終了することができます。即時シャットダウンでは、すべての CICS リスナーがただちに終了するよう指示されます。

SMSGSUP=NO|YES
YES を指定すると、メッセージ EZY1318E、EZY1325I、および EZY1330I が抑止されます。デフォルトは **NO** で、これらのメッセージの発行を許可します。

TCPADDR=SOCKET00|*nn*
z/VSE TCP/IP アドレス・スペースの名前。z/VSE は **SOCKET***nn* をサポートします。ここで *nn* は、接続に使用する TCP/IP スタックの ID です。

LISTENER
リスナー・オブジェクトを識別します。これは、リスナー・レコードを作成します。TYPE=LISTENER が指定される場合、以下のパラメーターが適用されます。

ACCTIME=60|*nn*
CICS リスナー・インターフェースのシャットダウンまたは CICS のシャットダウンをチェックする前に、リスナーが接続要求を待機する時間 (秒)。デフォルトは 60 です。この値に大きい値を設定すると、負荷の小さいシステムでは CPU 消費が最小化されますが、シャットダウン処理時間が長くなります。この値を低く設定すると、CPU の使用量は増大しますが、シャットダウン・プロセスが円滑に行われるようになります。

AF=INET|INET6
定義されているリスナーが IPv6 パートナーをサポートして IPv6 ソケット

記述子を IPv6 子サーバー・プログラムに与えられるようにするかどうかを指定します。INET6 は、リスナーが IPv6 ソケットを子サーバー・プログラムに与えることを示します。デフォルトの INET は、リスナーが IPv4 ソケットを子サーバー・プログラムに与えることを示します。
TAKESOCKET コマンドを実行する子サーバー・プログラムは、リスナーによって与えられるソケットのドメインと一致するようにします。

APPLID=*xxx*
このリスナーを定義する対象の CICS オブジェクトのアプリケーション ID。このパラメーターが省略された場合、前の TYPE=CICS マクロの APPLID が使用されます。

BACKLOG=20|*nn*
このリスナーの待ち行列に入れられる、受け入れられていない接続の数。デフォルト値は 20 です。

注: CSI International の TCP/IP for z/VSE スタックを使用する場合、バックログ機能のスタック・インプリメンテーションのため、常に 1 の値が使用されます。

CSDELAY=*hhmmss*
(拡張 CICS リスナーのみ) このパラメーターは **CSSTTYP** が IC の場合のみ適用できます。**EXEC CICS START** コマンドで使用される遅延間隔を指定します。フォーマットは *hhmmss* (時間/分/秒) です。

CSSTTYP=IC|KC|TD
(拡張 CICS リスナーのみ) このパラメーターは、子サーバー・タスクのデフォルトの始動方法を指定します。これは、セキュリティー/トランザクション出口によってオーバーライドすることができます。可能な値は、IC、KC、および、TD です。

IC 子サーバー・タスクが、遅延間隔として **CSDELAY** で指定した値 (またはセキュリティー/トランザクション出口でオーバーライドされた値) で、 **EXEC CICS START** を使用して始動することを示しています。

KC 子サーバー・タスクが、遅延間隔を指定せずに **EXEC CICS START** を使用して始動することを示します。これはデフォルトです。

TD 子サーバー・タスクが **EXEC CICS WRITEQ TD** コマンドを使用して始動することを示します。このコマンドは、一時データを使用して子サーバー・タスクを起動します。

CSTRAN=*xxx*
(拡張 CICS リスナーのみ) この必須パラメーターは、リスナーが始動するデフォルトの子サーバー・トランザクションを指定します。**CSTRAN** は、セキュリティー/トランザクション出口によってオーバーライドすることができます。リスナーが EZAO オペレーター・トランザクションによって始動されるときに、この子サーバー・トランザクションが CICS で定義済みであり使用可能であることが検査されます。

FORMAT=STANDARD|ENHANCED
デフォルトの **STANDARD** は、クライアントからの標準ヘッダーの送信を要求

する、オリジナルの CICS リスナーを示しています。**ENHANCED** は、クライアントからの標準ヘッダーを要求しない、拡張 CICS リスナーを示しています。

GIVTIME=*nn*
リスナーが **GIVESOCKET** への応答を待機する時間 (秒)。時間を過ぎると、リスナーは、サーバー・トランザクションが開始しなかったか、**TAKESOCKET** が失敗したかのどちらかだと想定します。リスナーは、サーバーが始動に失敗し、ソケット (接続) をクローズしたことを示すメッセージをクライアントに送信します。このパラメーターを指定しない場合、**ACCTIME** 値が使用されます。

IMMED=YES|NO
デフォルトの **YES**は、リスナーはインターフェースが始動する時に始動することを示します。 **NO** は、このリスナーが、EZAO トランザクションを使用して独自に始動することを示します。

MINMSGL=4|*nn*
(標準 CICS リスナーのみ) このパラメーターは、クライアントからリスナーへのトランザクション初期メッセージの最小長を指定します。デフォルト値は 4 です。リスナーは、この長さのデータの受信が終わるまで、接続を介した読み取りを続行します。

MSGFORM=ASCII|EBCDIC
(拡張 CICS リスナーのみ) このパラメーターは、クライアントに戻されるエラー・メッセージが ASCII か、または EBCDIC かを示します。デフォルトは ASCII です。MSGFORM は EZAC 画面に MSGFORMat として表示されます。

MSGLEN=0|*nnn*
(拡張 CICS リスナーのみ) このパラメーターは、クライアントから受け取るデータの長さを指定します。有効な範囲は 0 から 999 までです。値が 0 の場合は、リスナーはクライアントからデータを読み取りません。

NUMSOCK=50|*nnn*
リスナーがサポートするソケットの数。1 つのソケットは、listen するソケットです。他のソケットは、GIVESOCKET 呼び出しを使用して接続をサーバーに渡すために使用されます。アクティブにできる並行 GIVESOCKET 要求の最大数は、この値から 1 を引いた数です。デフォルト値は 50 です。

PORT=*nnnnn*
リスナーが接続を受け入れるのに使用するポート番号。このパラメーターは必須です。ポートは共用することもできます。指定可能な値は 1 から 65535 までです。

REATIME=*nn*
このリスナーが READ への応答を待機する時間 (秒)。この時間を過ぎると、リスナーは、クライアントが失敗したと想定し、ソケットをクローズして接続を終了します。このパラメーターを指定しない場合、読み取りタイムアウトの検査は行われません。

RTYTIME=15|*nnn*
TCP/IP スタックの停止が起きた後、接続または再接続を試行する前に、リ

スナーが待機する時間の長さを秒単位で指定します。**0** を指定すると、リスナーはすべてのリソースをクリーンアップし、終了します。0 より大きく 15 より小さい値は、15 秒の RTYTIME 値になります。リスナー・タスクは 15 秒の遅延後に接続または再接続を試行します。接続の試行先であるスタックは、リスナーの IP CICS ソケット・インターフェースの TCPADDR 構成オプションに指定されたスタックです。接続が失敗すると、リスナー・タスクは RTYTIME パラメーターに指定された時間の長さだけ遅延します。このインターバルが経過したあとで、リスナーはスタックへの接続を試行します。接続が成功するか、またはオペレーターがこの操作を終了するまで、リスナーはスタックへの接続を試行し続けます。*nnn* の値は 0 から 999 までです。デフォルト設定は、15 秒です。次の表では、RTYTIME 値とリスナーの TCP スタック状態との組み合わせに基づくリスナーのアクションについて、その要約を示しています。このパラメーターはオプションです。

表 *16. RTYTIME* とスタック状態に基づくリスナーのアクション

リスナー	**RTYTIME**	**TCP** 停止	**TCP** 稼働
最初に始動した	0	リスナーの終了	リスナーの初期設定
	>0	リスナーの待機	
以前にアクティブだった	0	リスナーの終了	
	>0	リスナーの待機	

SECEXIT
リスナーが使用するユーザー作成セキュリティー出口の名前。デフォルトでは、セキュリティー出口はありません。リスナーは EXEC CICS LINK コマンドを使用して、セキュリティー出口に制御を与えます。リスナーが EZAO オペレーター・トランザクションによって始動するときに、指定されたセキュリティー出口プログラムが CICS に対して定義されていて使用可能になっていることを確認するために、検査が行われます。

TRANID=EZAL|*xxx*
リスナーのトランザクション名。デフォルト値は EZAL です。

TRANTRN=YES|NO
(標準 CICS リスナーのみ) YES は、ユーザー・データの変換がトランザクション・コードの文字フォーマットに基づくことを示します。つまり、TRANTRN に YES を指定すると、TRANUSR が YES で、トランザクション・コードが大文字 EBCDIC でない場合にのみ、ユーザー・データが変換されます。 TRANTRN に NO が指定されている場合、TRANUSR が YES の場合にのみ、ユーザー・データが変換されます。TRANTRN のデフォルト値は YES です。

注**:** TRANTRN の指定内容にかかわらず、トランザクション・コードの変換は、最初の文字が大文字 EBCDIC でない場合に実行されます。

TRANUSR=YES|NO
(標準 CICS リスナーのみ) **NO** は、トランザクション初期メッセージのユーザー・データが ASCII から EBCDIC に変換されないことを示します。**YES** は、 535 ページの表 17 に示されるように、TRANTRN およびトランザ

クション・コードの文字フォーマットに基づいてユーザー・データが変換されることを示します。TRANUSR のデフォルトは YES です。

注: 以前のインプリメンテーションでは、TRANTRN および TRANUSR が両方とも YES に設定されているかのように機能しました。通常、インターネット上のデータは ASCII であり、変換が必要です。例外は、EBCDIC クライアントからのデータ、または、ユーザー・フィールド内のバイナリー・データです。これらの場合は、値に合わせた設定を行う必要があります。混合環境で操作している場合は、複数のポート上で複数のリスナーを使用します。

表 17. *Tranid* およびユーザー・データ変換の条件

TRANTRN	**TRANUSR**	**Tranid** フォーマット	**Tranid** の変換	ユーザー・データの変換
はい	はい	EBCDIC	いいえ	いいえ
はい	いいえ	EBCDIC	いいえ	いいえ
いいえ	はい	EBCDIC	いいえ	はい
いいえ	いいえ	EBCDIC	いいえ	いいえ
はい	はい	ASCII	はい	はい
はい	いいえ	ASCII	はい	いいえ
いいえ	はい	ASCII	はい	はい
いいえ	いいえ	ASCII	はい	いいえ

FINAL
生成の終わりを示します。サブパラメーターはありません。

構成データ・セットのカスタマイズ

CICS リスナー・サポートを使用し、構成ファイルによって制御される各 CICS ごとに、1 つの CICS オブジェクトがあります。CICS オブジェクトは、それが参照する CICS の APPLID で識別されます。

CICS に対して定義された各リスナーごとに 1 つのリスナー・オブジェクトがあります。CICS がリスナーを持たないことも可能ですが、これは一般的な手法ではありません。1 つの CICS が複数のリスナーを持つことができます。それらは、提供されたリスナーの、仕様が違う複数のインスタンスであるか、複数のユーザー作成リスナー、または任意の組み合わせです。

構成トランザクション (EZAC)

EZAC トランザクションは、パネル駆動型インターフェースであり、構成ファイルの追加、削除、または変更を行うことができます。以下の表に、EZAC トランザクションでサポートされる機能を示します。

コマンド	オブジェクト	関数
536 ページの『ALTER 機能』	CICS/リスナー	既存リソース定義の属性を変更する。

コマンド	オブジェクト	関数
540 ページの『COPY 機能』	CICS/リスナー	• CICS - CICS オブジェクトおよびそれに関連付けられたリスナーをコピーして、別の CICS オブジェクトを作成する。その新しい CICS オブジェクトが既に存在する場合、COPY は失敗します。 • リスナー - リスナー・オブジェクトをコピーして、別のリスナー・オブジェクトを作成する。その新しいリスナー・オブジェクトが既に存在する場合、COPY は失敗します。
541 ページの『CONVERT 機能』	リスナー	標準リスナーから拡張リスナーへの変換、またはその逆の変換を行う。
543 ページの『DEFINE 機能』	CICS/リスナー	新規リソース定義を作成する。
547 ページの『DELETE 機能』	CICS/リスナー	• CICS - CICS オブジェクトおよびそれに関連付けられたすべてのリスナーを削除する。 • リスナー - リスナー・オブジェクトを削除する。
549 ページの『DISPLAY 機能』	CICS/リスナー	CICS/リスナー・オブジェクトに対して指定されたパラメーターを表示する。
551 ページの『RENAME 機能』	CICS/リスナー	COPY を実行した後、元のオブジェクトの DELETE を実行する。

EZAC を入力すると、次の画面が表示されます。

```
EZAC                                                    APPLID=DBDCCICS
 ENTER ONE OF THE FOLLOWING
ALter
Convert
COpy
DEFine
DELete
DISplay
REName









PF 1 HELP       3 END            6 CRSR            9 MSG            12 CNCL
```

ALTER 機能

ALTER 機能は、CICS オブジェクトまたはそれらのリスナー・オブジェクト、あるいは両方を変更するのに使用します。

EZAC 初期画面で ALter を指定するか、空白画面で EZAC AL を入力すると、次の画面が表示されます。

```
EZAC,ALTER,                                                  APPLID=DBDCCICS

 Enter One of the Following

 CICS
 LISTENER

 PF 3 END                                                        12 CNCL
```

ファースト・パス: EZAC ALTER CICS または EZAC ALTER LISTENER を入力すると、これをショートカットできます。

ALTER CICS

CICS オブジェクトを変更する場合、次の画面が表示されます。

```
EZAC,ALTER,CICS                                              APPLID=DBDCCICS
  ENTER ALL FIELDS

 APPLID       ===>                        APPLID of CICS System

 PF 3 END                                                        12 CNCL
```

APPLID を入力すると、次の画面が表示されます。

```
EZAC,ALTER,CICS                                        APPLID = xxxxxxxx

OVERTYPE TO ENTER

APPLID       ===> XXXXXXXX             APPLID of CICS System
TCPADDR      ===> xxxxxxxx             Name of TCP Address Space
CACHMIN      ===> xxx                  Minimum Refresh Time for Cache
CACHMAX      ===> xxx                  Maximum Refresh Time for Cache
CACHRES      ===> xxx                  Maximum number of Resolvers
ERRORTD      ===> xxxx                 TD Queue for Error Messages
SMSGSUP      ===> xxx                  Suppress Task Started Messages
PLTSDI       ===> xxx                  CICS PLT Shutdown Immediately









Press enter to confirm function function
or PF12 to make more changes

PF 3 END                                                          12 CNCL
```

表示された値の確認がシステムから要求されます。変更が確認されると、CICS ソケット・インターフェースを次に初期設定したときに、変更した値が有効になります。

ALTER LISTENER

リスナー・オブジェクトを変更する場合、次の画面が表示されます。

```
EZAC,ALTER,                                                 APPLID=DBDCCICS
  ENTER ALL FIELDS

 APPLID       ===>                       APPLID of CICS System
 NAME         ===>                       Transaction Name of Listener

                                                           APPLID=DBDCCICS

 PF 3 END                                                        12 CNCL
```

名前を入力すると、次の画面が表示されます。

注: 拡張リスナー用の画面 1 と標準リスナー用の画面 1 は似ています。

```
EZAC,ALTER,LISTENER (standard listener.  screen 1 of 2) APPLID = xxxxxxxx

 OVERTYPE TO ENTER

 APPLID       ===> xxxxxxxx             APPLID of CICS System
 TRANID       ===> xxxxx                Transaction Name of Listener
 PORT         ===> xxxxx                Port Number of Listener
 AF           ===> xxxxx                Listener Address Family
 IMMEDIATE    ===> xxx                  Immediate Startup   Yes|No
 BACKLOG      ===> xxx                  Backlog Value for Listener
 NUMSOCK      ===> xxx                  Number of Sockets in Listener
 ACCTIME      ===> xxx                  Timeout Value for ACCEPT
 GIVTIME      ===> xxx                  Timeout Value for GIVESOCKET
 REATIME      ===> xxx                  Timeout Value for READ
 RTYTIME      ===> xxx                  Stack Connection Retry Time




 Verify parameters, press PF8 to go to screen 2


 PF 3 END                               8 NEXT                      12 CNCL
```

次の画面 2 は標準リスナー用に表示されたものです。

```
EZAC,ALTER,LISTENER (standard listener.  screen 2 of 2)  APPLID = xxxxxxxx

 Overtype to Enter

 MINMSGL      ===> xxx                  Minimum Message Length
 TRANTRN      ===> xxx                  Translate TRNID     Yes|No
 TRANUSR      ===> xxx                  Translate User Data Yes|No
 SECEXIT      ===> xxxxxxxx             Name of Security Exit

 Verify parameters, press PF7 to go back to screen 1
                  or ENTER if finished making changes


 PF 3 END                7 PREV                                  12 CNCL
```

次の画面 2 は拡張リスナー用に表示されたものです。

```
EZAC,ALTER,LISTENER (enhanced listener.  screen 2 of 2)  APPLID = xxxxxxxx

 Overtype to Enter

 CSTRANid     ===> XXXX                Child Server Transaction Name
 CSSTTYPe     ===> xx                  Startup Method (KC|IC|TD)
 CSDELAY      ===> xxxxxx              Delay Interval (hhmmss)
 MSGLENgth    ===> xxx                 Message Length (0-999)
 MSGFORMat    ===> xxxxx               Enter ASCII|EBCDIC
 USEREXIT     ===>                     Name of User/Security exit









 Verify parameters, press PF7 to go back to screen 1
                  or ENTER if finished making changes


 PF 3 END                 7 PREV                                  12 CNCL
```

表示された値の確認がシステムから要求されます。変更が確認されると、CICS ソケット・インターフェースを次に初期設定したときに、変更した値が有効になります。

COPY 機能

COPY 機能は、オブジェクトを新規オブジェクトにコピーするのに使用します。

EZAC 初期画面で COpy を指定するか、空白画面で EZAC CO を入力すると、次の画面が表示されます。

```
EZAC,COPY,                                                  APPLID=DBDCCICS

 Enter One of the Following

 CICS
 LISTENER
```

ファースト・パス: EZAC COPY CICS または EZAC COPY LISTENER を入力すると、これをショートカットできます。

COPY CICS

前の画面で CICS を入力した場合、次の画面が表示されます。

```
EZAC,COPY,                                                  APPLID=DBDCCICS
  ENTER ALL FIELDS
 SCICS        ===> ........          APPLID of Source CICS
 TCICS        ===> ........          APPLID of Target CICS









 PF 3 END                                      9 MSG           12 CNCL
```

ソース CICS オブジェクトおよびターゲット CICS オブジェクトの APPLID を入力すると、確認が要求されます。確認を入力すると、コピーが実行されます。

COPY LISTENER

COPY LISTENER を指定すると、次の画面が表示されます。

```
EZAC,COPY,                                                  APPLID=DBDCCICS
  ENTER ALL FIELDS
 SCICS        ===> ........          APPLID of Source CICS
 SLISTener    ===> ....              Transaction Name of Source Listener
 TCICS        ===> ........          APPLID of Target CICS
 TLISTener    ===> ....              Transaction Name of Target Listener









 PF 3 END                                                    12 CNCL
```

ソースおよびターゲット CICS オブジェクトの APPLID と、ソースおよびターゲット・リスナーの名前を入力すると、確認が要求されます。確認を入力すると、コピーが実行されます。

CONVERT 機能

CONVERT 機能は標準リスナーを拡張リスナーに変換するために使用します。

EZAC 初期画面で CONVERT を指定するか、空白画面で EZAC CONVERT を入力すると、次の画面が表示されます。

```
EZAC,CONVERT,LISTENER                                    APPLID = xxxxxxxx

OVERTYPE TO ENTER

APPLID       ===> xxxxxxxx            APPLID of CICS System

TRANID       ===> xxxx                Transaction Name of Listener

Format       ===> xxxxxxxx            Enter STANDARD|ENHANCED









PF 3 END                                                          12 CNCL
```

名前を入力すると、次の画面が表示されます。

注: 拡張リスナー用の画面 1 と標準リスナー用の画面 1 は似ています。

```
EZAC,CONVERT,LISTENER (standard listener.  screen 1 of 2) APPLID = xxxxxxxx

 OVERTYPE TO ENTER

 APPLID       ===> xxxxxxxx            APPLID of CICS System
 TRANID       ===> xxxxx               Transaction Name of Listener
 PORT         ===> xxxxx               Port Number of Listener
 AF           ===> xxxxx               Listener Address Family
 IMMEDIATE    ===> xxx                 Immediate Startup   Yes|No
 BACKLOG      ===> xxx                 Backlog Value for Listener
 NUMSOCK      ===> xxx                 Number of Sockets in Listener
 ACCTIME      ===> xxx                 Timeout Value for ACCEPT
 GIVTIME      ===> xxx                 Timeout Value for GIVESOCKET
 REATIME      ===> xxx                 Timeout Value for READ
 RTYTIME      ===> xxx                 Stack Connection Retry Time




 Verify parameters, press PF8 to go to screen 2


 PF 3 END                              8 NEXT                      12 CNCL
```

次の画面 2 は標準リスナー用に表示されたものです。

```
EZAC,CONVERT,LISTENER (standard listener.  screen 2 of 2)  APPLID = xxxxxxxx

 Overtype to Enter

 MINMSGL      ===> xxx                  Minimum Message Length
 TRANTRN      ===> xxx                  Translate TRNID     Yes|No
 TRANUSR      ===> xxx                  Translate User Data Yes|No
 SECEXIT      ===> xxxxxxxx             Name of Security Exit









 Verify parameters, press PF7 to go back to screen 1
                  or ENTER if finished making changes


 PF 3 END                 7 PREV                                  12 CNCL
```

次の画面 2 は拡張リスナー用に表示されたものです。

```
EZAC,CONVERT,LISTENER (enhanced listener.  screen 2 of 2)  APPLID = xxxxxxxx

 Overtype to Enter

 CSTRANid     ===> XXXX                 Child Server Transaction Name
 CSSTTYPe     ===> xx                   Startup Method (KC|IC|TD)
 CSDELAY      ===> xxxxxx               Delay Interval (hhmmss)
 MSGLENgth    ===> xxx                  Message Length (0-999)
 MSGFORMat    ===> xxxxx                Enter ASCII|EBCDIC
 USEREXIT     ===>                      Name of User/Security exit









 Verify parameters, press PF7 to go back to screen 1
                  or ENTER if finished making changes


 PF 3 END                 7 PREV                                  12 CNCL
```

定義を入力すると、確認が要求されます。確認を入力すると、構成ファイルにオブジェクトが作成されます。

DEFINE 機能

DEFINE 機能は、CICS オブジェクトおよびそれらのリスナー・オブジェクトを作成するのに使用します。

EZAC 初期画面で DEFine を指定するか、空白画面で EZAC DEF を入力すると、次の画面が表示されます。

```
EZAC,DEFINE,                                                APPLID=DBDCCICS

 Enter One of the Following

 CICS
 LISTENER









 PF 3 END                                                        12 CNCL
```

ファースト・パス: EZAC DEFINE CICS または EZAC DEFINE LISTENER を入力すると、これをショートカットできます。

DEFINE CICS

CICS オブジェクトを定義する場合、次の画面が表示されます。

```
EZAC,DEFINE,CICS                                            APPLID=DBDCCICS
  ENTER ALL FIELDS

 APPLID       ===>                        APPLID of CICS System









 PF 3 END                                                        12 CNCL
```

APPLID を入力すると、次の画面が表示されます。

```
EZAC,DEFINE,CICS                                        APPLID = xxxxxxxx

OVERTYPE TO ENTER

APPLID       ===> XXXXXXXX            APPLID of CICS System
TCPADDR      ===> xxxxxxxx            Name of TCP Address Space
CACHMIN      ===> xxx                 Minimum Refresh Time for Cache
CACHMAX      ===> xxx                 Maximum Refresh Time for Cache
CACHRES      ===> xxx                 Maximum number of Resolvers
ERRORTD      ===> xxxx                TD Queue for Error Messages
SMSGSUP      ===> xxx                 Suppress Task Started Messages
PLTSDI       ===> xxx                 CICS PLT Shutdown Immediately









Press enter to confirm function function
or PF12 to make more changes

PF 3 END                                                          12 CNCL
```

定義を入力すると、確認が要求されます。確認を入力すると、構成ファイルにオブジェクトが作成されます。

DEFINE LISTENER

リスナー・オブジェクトを定義する場合、次の画面が表示されます。

```
EZAC,DEFINE,LISTENER                                    APPLID = xxxxxxxx

OVERTYPE TO ENTER

APPLID       ===> xxxxxxxx            APPLID of CICS System

TRANID       ===> xxxx                Transaction Name of Listener

Format       ===> xxxxxxxx            Enter STANDARD|ENHANCED









PF 3 END                                                          12 CNCL
```

名前を入力すると、次の画面が表示されます。

注: 拡張リスナー用の画面 1 と標準リスナー用の画面 1 は似ています。

```
EZAC,DEFINE,LISTENER (standard listener.  screen 1 of 2) APPLID = xxxxxxxx

 OVERTYPE TO ENTER

 APPLID       ===> xxxxxxxx             APPLID of CICS System
 TRANID       ===> xxxxx                Transaction Name of Listener
 PORT         ===> xxxxx                Port Number of Listener
 AF           ===> xxxxx                Listener Address Family
 IMMEDIATE    ===> xxx                  Immediate Startup   Yes|No
 BACKLOG      ===> xxx                  Backlog Value for Listener
 NUMSOCK      ===> xxx                  Number of Sockets in Listener
 ACCTIME      ===> xxx                  Timeout Value for ACCEPT
 GIVTIME      ===> xxx                  Timeout Value for GIVESOCKET
 REATIME      ===> xxx                  Timeout Value for READ
 RTYTIME      ===> xxx                  Stack Connection Retry Time




 Verify parameters, press PF8 to go to screen 2


 PF 3 END                               8 NEXT                      12 CNCL
```

次の画面 2 は標準リスナー用に表示されたものです。

```
EZAC,DEFINE,LISTENER (standard listener.  screen 2 of 2)  APPLID = xxxxxxxx

 Overtype to Enter

 MINMSGL      ===> xxx                  Minimum Message Length
 TRANTRN      ===> xxx                  Translate TRNID     Yes|No
 TRANUSR      ===> xxx                  Translate User Data Yes|No
 SECEXIT      ===> xxxxxxxx             Name of Security Exit









 Verify parameters, press PF7 to go back to screen 1
                  or ENTER if finished making changes


 PF 3 END                7 PREV                                  12 CNCL
```

次の画面 2 は拡張リスナー用に表示されたものです。

```
EZAC,DEFINE,LISTENER (enhanced listener.  screen 2 of 2)  APPLID = xxxxxxxx

 Overtype to Enter

 CSTRANid     ===> XXXX                Child Server Transaction Name
 CSSTTYPe     ===> xx                  Startup Method (KC|IC|TD)
 CSDELAY      ===> xxxxxx              Delay Interval (hhmmss)
 MSGLENgth    ===> xxx                 Message Length (0-999)
 MSGFORMat    ===> xxxxx               Enter ASCII|EBCDIC
 USEREXIT     ===>                     Name of User/Security exit









 Verify parameters, press PF7 to go back to screen 1
                  or ENTER if finished making changes


 PF 3 END                 7 PREV                                  12 CNCL
```

定義を入力すると、確認が要求されます。確認を入力すると、構成ファイルにオブジェクトが作成されます。

DELETE 機能

DELETE 機能は、CICS オブジェクトまたはリスナー・オブジェクトを削除するのに使用します。

ある CICS オブジェクトを削除すると、その CICS オブジェクト内のすべてのリスナー・オブジェクトが削除されます。EZAC 初期画面で DELete を指定するか、空白画面で EZAC DEL を入力すると、次の画面が表示されます。

```
EZAC,DELETE,                                                 APPLID=DBDCCICS

 Enter One of the Following

 CICS
 LISTENER









 PF 3 END                                                        12 CNCL
```

DELETE CICS

DELETE CICS を指定すると、次の画面が表示されます。

```
EZAC,DELETE,CICS                                          APPLID=DBDCCICS
  ENTER ALL FIELDS

 APPLID       ===>                        APPLID of CICS System









 PF             3 END                                          12 CNCL
```

APPLID を入力すると、確認が要求されます。確認を入力すると、CICS オブジェクトが削除されます。

DELETE LISTENER

DELETE LISTENER を指定すると、次の画面が表示されます。

```
EZAC,DELETE,LISTENER                                      APPLID=DBDCCICS
  ENTER ALL FIELDS

 APPLID       ===>                        APPLID of CICS System
 NAME         ===>                        Transaction Name of Listener









 PF             3 END                                          12 CNCL
```

APPLID とリスナー名を入力すると、確認が要求されます。確認を入力すると、リスナー・オブジェクトが削除されます。

DISPLAY 機能

DISPLAY 機能は、オブジェクトの仕様を表示するのに使用します。

EZAC 初期画面で DISplay を指定するか、空白画面で EZAC DIS を入力すると、次の画面が表示されます。

```
EZAC,DISPLAY,                                                  APPLID=DBDCCICS

 Enter One of the Following

 CICS
 LISTENER









 PF 3 END                                                          12 CNCL
```

ファースト・パス: EZAC DISPLAY CICS または EZAC DISPLAY LISTENER を入力すると、これをショートカットできます。

DISPLAY CICS

DISPLAY CICS を指定すると、次の画面が表示されます。

```
EZAC,DISPLAY,                                                  APPLID=DBDCCICS
  ENTER ALL FIELDS

 APPLID       ===>                        APPLID of CICS System









 PF             3 END                                              12 CNCL
```

APPLID を入力すると、次の画面が表示されます。

```
EZAC,DISplay,CICS                                    APPLID = xxxxxxxx

OVERTYPE TO ENTER

APPLID       ===> XXXXXXXX            APPLID of CICS System
TCPADDR      ===> xxxxxxxx            Name of TCP Address Space
CACHMIN      ===> xxx                 Minimum Refresh Time for Cache
CACHMAX      ===> xxx                 Maximum Refresh Time for Cache
CACHRES      ===> xxx                 Maximum number of Resolvers
ERRORTD      ===> xxxx                TD Queue for Error Messages
SMSGSUP      ===> xxx                 Suppress Task Started Messages
PLTSDI       ===> xxx                 CICS PLT Shutdown Immediately









Press ENTER or PF3 to exit

PF 3 END                                                    12 CNCL
```

DISPLAY LISTENER

DISPLAY LISTENER を指定すると、次の画面が表示されます。

```
EZAC,DISPLAY,                                              APPLID=DBDCCICS
  ENTER ALL FIELDS

 APPLID       ===>                      APPLID of CICS System
 NAME         ===>                      Transaction Name of Listener









 PF 3 END                                                    12 CNCL
```

APPLID と名前を入力すると、次の画面が表示されます。

```
EZAC,DISPLAY,LISTENER                                          APPLID=DBDCCICS


 APPLID       ===> ........          APPLID of CICS System
 TRanname     ===> ........          Transaction Name of Listener
 POrt         ===> .....             Port Number of Listener
 IMMediate    ===> Yes               Immediate Startup        Yes|No
 BAcklog      ===> 020               Backlog Value for Listener
 NUMsock      ===> 50                Number of Sockets in Listener
 MINmsgl      ===> 04                Minimum Message Length
 ACCTime      ===> 60                Timeout Value for Accept
 GIVTime      ===> 60                Timeout Value for Givesocket
 REATime      ===> 10                Timeout Value for Read
 FASTread     ===> Yes               Read immediately         Yes|No
 TRANTrn      ===> Yes               Translate Trans. Name    Yes|No
 TRANUsr      ===> Yes               Translate User Data      Yes|No
 SECexit      ===> ........          Security Exit Name



 PF 3 END                                                        12 CNCL
```

RENAME 機能

RENAME 機能は、CICS オブジェクトまたはリスナー・オブジェクトの名前を変更するのに使用します。

これは、COPY 操作と、その後のソース・オブジェクトの DELETE からなります。CICS オブジェクトに対して実行すると、そのオブジェクトと、それに関連付けられたすべてのリスナーが名前変更されます。リスナー・オブジェクトに対して実行すると、そのリスナーだけが名前変更されます。

EZAC 初期画面で RENname を指定するか、空白画面で EZAC REN を入力すると、次の画面が表示されます。

```
EZAC,RENAME,                                                   APPLID=DBDCCICS

 Enter One of the Following

 CICS
 LISTENER

 PF 3 END                                                        12 CNCL
```

ファースト・パス: EZAC RENAME CICS または EZAC RENAME LISTENER を入力すると、これをショートカットできます。

RENAME CICS

前の画面で CICS を入力した場合、次の画面が表示されます。

```
EZAC,RENAME,                                                APPLID=DBDCCICS
  ENTER ALL FIELDS

 SCICS        ===> ........          APPLID of Source CICS
 TCICS        ===> ........          APPLID of Target CICS









 PF 3 END                                      9 MSG          12 CNCL
```

ソース CICS オブジェクトおよびターゲット CICS オブジェクトの APPLID を入力すると、確認が要求されます。確認を入力すると、名前変更が実行されます。

RENAME LISTENER

RENAME LISTENER を指定すると、次の画面が表示されます。

```
EZAC,RENAME,                                                APPLID=DBDCCICS
  ENTER ALL FIELDS

 SCICS        ===> ........          APPLID of Source CICS
 SLISTener    ===> ....              Transaction Name of Source Listener
 TCICS        ===> ........          APPLID of Target CICS
 TLISTener    ===> ....              Transaction Name of Target Listener









 PF 3 END                                                    12 CNCL
```

ソースおよびターゲット CICS オブジェクトの APPLID と、ソースおよびターゲット・リスナーの名前を入力すると、確認が要求されます。確認を入力すると、名前変更が実行されます。

第 19 章 CICS ドメイン・ネーム・サーバー・キャッシュの構成

ドメイン・ネーム・サーバー・キャッシュの概要

ドメイン・ネーム・サーバー (DNS) は、個人の名前や、住所、電話番号が記載されている電話帳のようなものです。ネーム・サーバーは、ホスト名を IP アドレスに、または IP アドレスをホスト名にマップします。ネーム・サーバーは、各ホストについて、IP アドレス、ニックネーム、メール情報、および使用可能なウェルノウン・サービス (例えば、SMTP、FTP、Telnet など) を保持しています。

ホスト名から IP アドレスへの変換は、DNS の 1 つの使用法にすぎません。他のタイプのホスト関連情報も、保管および照会することができます。ネーム・サーバーのリソース・レコードへの入力データを介して、さまざまなタイプの情報が定義されます。

CICS ドメイン・ネーム・サーバー・キャッシュ機能はオプションの機能ですが、高度にアクティブな CICS クライアント環境では役立ちます。この機能は、TCP/IP for z/VSE においてサポートされている gethostbyname() 呼び出しを、gethostbyname() の結果を将来の使用のために保管するキャッシュと結合します。同じドメイン・ネームの集合に関する要求をシステムが繰り返し受け取るような場合、DNS を使用するとパフォーマンスが大幅に向上します。

機能のコンポーネント

この機能は、次の 3 つのパーツで構成されます。

- キャッシュ用に使用される VSAM ファイル。
- キャッシュ・ファイルを初期設定するのに使用される EZACICR マクロ。
- gethostbyname ソケット呼び出しの代わりに CICS アプリケーションによって起動される、CICS アプリケーション・プログラム EZACIC25。

VSAM キャッシュ・ファイル

キャッシュ・ファイルは、右に 2 進ゼロを埋めたホスト名をキーにした、VSAM KSDS (キー順データ・セット) です。キャッシュ・レコードには、ドメイン・ネーム・サーバーから戻された圧縮バージョンの hostent 構造体と、最終リフレッシュ時刻フィールドが含まれています。レコードが取得されると、EZACIC25 は、そのレコードが使用可能かどうかを、現在時刻と最終リフレッシュ時刻の差に基づいて判断します。

EZACICR マクロ

EZACICR マクロは、キャッシュ・ファイル用の初期設定モジュールを構築します。キャッシュ・ファイルには始めから少なくとも 1 つのレコードが含まれていないと、EZACIC25 モジュールでの更新ができないためです。パフォーマンスを最適化するため、頻繁な使用が予期されるホストについて、それらのホスト名を「ダミー」のレコードとしてプリロードすることもできます。こうすると、ファイルがよりコンパクトになり、キャッシュを必要とする入出力が最小化されます。1 つもダ

ミー・レコードを指定しない場合、このマクロが、2 進ゼロの 1 つのレコードを作成します。 555 ページの『ステップ 1: 初期設定モジュールの作成』を参照してください。

EZACIC25 モジュール

このモジュールは、通常の CICS アプリケーション・プログラムであり、EXEC CICS LINK コマンドで起動されます。起動元の CICS プログラムとこの DNS モジュール間の情報の受け渡しは、COMMAREA を介して行われます。ドメイン・ネームの解決が成功すると、EZACIC25 は、CICS からストレージを取得し、そのストレージ内で hostent 構造体を作成します。hostent 構造体に関する作業が終わったら、EXEC CICS FREEMAIN コマンドを使用して、このストレージを解放してください。

EZACIC25 モジュールは、4 個のパラメーターと、起動元アプリケーションから渡された情報を使用して、キャッシュを管理します。これらのパラメーターは、次のとおりです。

エラー宛先
: エラー・メッセージが送信される一時データ宛先。

最小リフレッシュ時間
: キャッシュ・レコードがリフレッシュされる最小間隔 (分)。キャッシュ・レコードがこの時間よりも「新しい」場合、そのレコードが使用されます。この値は、15 (分) に設定されます。

最大リフレッシュ時間
: キャッシュ・レコードがリフレッシュされる最大間隔 (分)。キャッシュ・レコードがこの時間よりも「古い」場合、そのレコードはリフレッシュされます。この値は、30 (分) に設定されます。

最大リゾルバー要求数
: リゾルバーへの最大同時要求数。これは 10 に設定されます。『DNS キャッシュの要求処理手順』を参照してください。

DNS キャッシュの要求処理手順

受け取った要求にキャッシュの取得が指定されている場合、以下の処理が行われます。

1. キャッシュからのこの項目の取得を試します。成功しなかった場合、要求がキャッシュのみを指定しているのでなければ、gethostbyname を発行します。
2. キャッシュからの取得が成功した場合、そのレコードの「経過時間」(現在時刻と、このレコードが作成またはリフレッシュされた時刻の差) を計算します。
   - 経過時間がキャッシュの最小リフレッシュ時間以下の場合、そのキャッシュ情報を使用し、要求元のために Hostent 構造体を構築します。その後、要求元に戻ります。
   - 経過時間がキャッシュの最大リフレッシュ時間よりも大きい場合、gethostbyname 呼び出しを発行し、その結果を使用してキャッシュ・レコードをリフレッシュします。
   - 経過時間がキャッシュの最小リフレッシュ時間と最大リフレッシュ時間の間にある場合、以下を行います。

a. キャッシュの最小リフレッシュ時間と最大リフレッシュ時間の差を計算し、それを最大同時リゾルバー要求数で除算します。この結果を時間増分と呼びます。
b. 時間増分に、現在アクティブなリゾルバー要求の数を乗算します。この時間を最小リフレッシュ時間に加算して、調整済みリフレッシュ時間を算出します。
c. レコードの経過時間が調整済みリフレッシュ時間よりも小さい場合、そのキャッシュ・レコードを使用します。
d. レコードの経過時間が調整済みリフレッシュ時間よりも大きい場合、gethostbyname 呼び出しを発行し、その結果を使用してキャッシュ・レコードをリフレッシュします。

- gethostbyname が発行され、成功した場合、キャッシュは更新され、その項目の更新時刻は現在時刻に変更されます。

DNS キャッシュの使用

DNS キャッシュを使用するには、3 つのステップが必要です。

1. 初期設定モジュールを作成します。そのモジュールが、ファイルおよび EZACIC25 モジュールを定義し、初期設定します。『ステップ 1: 初期設定モジュールの作成』を参照してください。
2. キャッシュ・ファイルを CICS に対して定義します。 557 ページの『ステップ 2: CICS へのキャッシュ・ファイルの定義』を参照してください。
3. EZACIC25 を使用して、CICS アプリケーション・モジュール内の **gethostbyname** 呼び出しを置き換えます。 557 ページの『ステップ 3: EZACIC25 の実行』を参照してください。

ステップ 1: 初期設定モジュールの作成

初期設定モジュールは、EZACICR マクロを使用して作成されます。このマクロは最低限 2 回起動されるようにコーディングされ、アセンブルされます。アセンブルされると、モジュールが生成されます。以下に例を示します。

```
EZACICR TYPE=INITIAL
EZACICR TYPE=FINAL
```

この例では、2 進ゼロのレコードを 1 つ作成する初期設定モジュールが生成されます。頻繁に使用されるドメイン・ネームのダミー・レコードをファイルにプリロードしたい場合は、次のようになります。

```
EZACICR TYPE=INITIAL
EZACICR TYPE=RECORD,NAME=HOSTA
EZACICR TYPE=RECORD,NAME=HOSTB
EZACICR TYPE=RECORD,NAME=HOSTC
EZACICR TYPE=FINAL
```

ここで、HOSTA、HOSTB、および HOSTC は、ダミー・レコードに入れるホスト名です。これらの名前は、どのような順序で指定してもかまいません。

パラメーター

EZACICR マクロの仕様は、次のとおりです。

TYPE 許容値は次の 3 つです。

INITIAL
生成入力の始まりを示します。この値は、入力ストリーム中に一度だけ先頭項目として指定されなければなりません。

RECORD
ユーザーが生成するダミー・レコードを示します。生成できるダミー・レコードの数は、0 から 4096 個であり、各レコードがそれぞれ固有の名前を持っていなければなりません。頻繁に使用されるホスト名についてダミー・レコードを生成すると、キャッシュ・ファイルのパフォーマンスが向上します。TYPE=RECORD ステートメントの前に、TYPE=INITIAL がなければなりません。

FINAL
生成入力の終わりを示します。この値は、入力ストリーム中に一度だけ最終項目として指定されなければなりません。TYPE=FINAL ステートメントの前に、TYPE=INITIAL がなければなりません。

AVGREC
平均キャッシュ・レコードの長さ。この値は TYPE=INITIAL マクロに指定され、デフォルト値は 500 です。適切な値を決定するための十分な統計が得られるまでは、デフォルト値を使用することをお勧めします。このパラメーターは、IDCAMS DEFINE ステートメントの RECORDSIZE パラメーター中の先頭サブパラメーターと同じです。ダミー・レコードの使用に加えて、このパラメーターを的確に定義すると、キャッシュ・ファイル内のコントロール・インターバルと制御域分割を最小化できます。

NAME
ダミー・レコードのホスト名を指定します。名前は、1 バイトから 255 バイトまでの長さでなければなりません。TYPE=RECORD 項目には、NAME オペランドが必須です。

z/VSE では、DNS キャッシュ・ファイルは事前定義済みですが、空です。VSAM クラスター VSE.EZACICS.CACHE として、カタログ VSESP.USER.CATALOG 内に定義されています。ファイル名は EZACACH です。このファイルの最小限の初期設定には、次の JCL を使用できます。

```
// JOB   CACHCRE
// EXEC  ASSEMBLY,GO
          EZACICR TYPE=INITIAL
          EZACICR TYPE=FINAL
          END
/*
/&
```

このジョブの実行時には EZACACH ファイルはクローズされている必要があることに注意してください。

作成後のキャッシュ・ファイルのレイアウトは、次のとおりです。

ホスト名
255 バイトの文字フィールドであり、ホスト名を指定します。このフィールドは、ファイルのキーです。

レコード・タイプ
: 1 バイトの 2 進数フィールドであり、レコード・タイプを指定します。この値は X'00000001' です。

最終リフレッシュ時刻
: 4 バイトのフィールドであり、最後のリフレッシュ時刻を指定します。この値は、1990 年 1 月 1 日 0000 時からの秒数を表し、EXEC CICS ASKTIME から取得した ABSTIME 値から 1990 年 1 月 1 日に対応する値を減算することで算出されます。

別名項目数
: ハーフワードの 2 進数フィールドであり、別名配列に含まれる項目の数を指定します。

別名配列リストのオフセット
: ハーフワードの 2 進数フィールドであり、レコード内での別名配列のオフセットを指定します。別名配列は、それぞれの後ろに x '00' バイトを付加した別名から構成されます。

INET アドレス数
: ハーフワードの 2 進数フィールドであり、レコード中の INET アドレスの数を指定します。

INET アドレス
: 1 つ以上のフルワード 2 進数フィールドであり、**gethostbyname()** から戻される INET アドレスを指定します。

別名
: 可変長の文字フィールドからなる配列であり、ドメイン・ネーム・サーバー・キャッシュから戻される別名を指定します。これらのフィールドは、2 進ゼロの 1 バイトで区切られます。これらのフィールドのそれぞれが、最大長 255 バイトです。

ステップ 2: CICS へのキャッシュ・ファイルの定義

この機能を CICS システムに追加するのに必要なすべての CICS 定義は、既に z/VSE に組み込まれています。

その中には、ファイル EZACACH 用の定義も、プログラム EZACIC25 用の定義も含まれています。

ステップ 3: EZACIC25 の実行

EZACIC25 は、gethostbyname ソケット呼び出しを置き換えるものです。これは、EXEC CICS LINK PROGRAM (EZACIC25) COMMAREA(com-area) によって起動されます。ここで、com-area は次のように定義されます。

戻りコード
: フルワードの 2 進数であり、この機能の結果を次のように指定します。

  -1 gethostbyname() 呼び出しから戻された ERRNO 値。ERRNO フィールドをチェックしてください。

  0 ホスト名をキャッシュ内で解決できなかったか、gethostbyname 呼び出しを使用して解決できなかったかのいずれかです。

  1 ホスト名は、キャッシュを使用して解決されました。

2 ホスト名は、gethostbyname 呼び出しを使用して解決されました。

ERRNO
フルワード 2 進数フィールドであり、GETHOSTBYNAME 呼び出しから戻された ERRNO を指定します。

HOSTENT アドレス
戻された HOSTENT 構造体のアドレス。

コマンド
4 バイトの文字フィールドであり、要求された操作を指定します。

GHBN
gethostbyname。これは、サポートされている唯一の関数です。

Namelen
フルワードの 2 進数変数であり、照会用にホスト名の実際の長さを指定します。

照会タイプ
1 バイトの文字フィールドであり、照会のタイプを次のように指定します。

0 キャッシュを使用して照会を試みます。成功しなかった場合、gethostbyname() 呼び出しの使用を試みます。

1 gethostbyname() 呼び出しを使用して照会を試みます。これを指定すると、キャッシュのこの項目が強制的にリフレッシュされます。

2 キャッシュのみを使用して照会を試みます。

注**:** 一致するレコードがキャッシュに入っている場合、そのレコードの内容が、レコードの経過時間に関係なく戻されます。

名前 256 バイトの文字変数であり、照会用にホスト名を指定します。

HOSTENT 構造体

戻される HOSTENT 構造体を 559 ページの図 52 に示します。

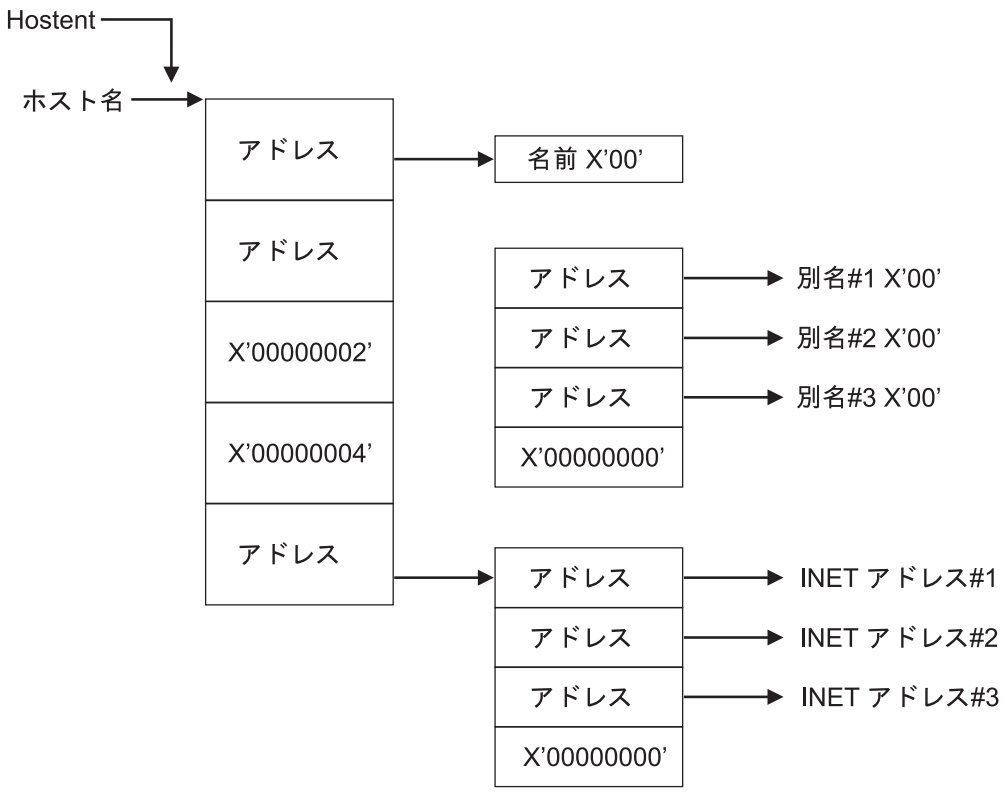


図 *52. DNS Hostent*

第 20 章 CICS リスナー・サポートの開始と停止

概説

このセクションでは、CICS リスナー・サポートの開始および停止 (使用可能化および使用不可化) の方法について説明します。以下の項目についての説明があります。

- CICS リスナー・サポートが自動的に開始および停止するように、システムをカスタマイズできます。『CICS リスナー・サポートの自動的な開始/停止』を参照してください。
- オペレーターは、CICS が初期設定された後、CICS リスナー・サポートを手動で開始および停止することもできます。『CICS リスナー・サポートの手動での開始/停止』を参照してください。
- CICS リスナー・サポートを CICS アプリケーション・プログラムから開始および停止することもできます。 566 ページの『プログラム・リンクを使用した CICS リスナー・サポートの開始/停止』を参照してください。

このセクションでは、上記の 3 つの方法をすべて説明します。

注: どのリスナーの開始よりも前に、リスナー・インターフェースが開始されなければなりません。リスナー・トランザクションは、トランザクション EZAO を介して、またはプログラム EZACIC20 を介して開始される必要があります。

CICS リスナー・サポートの自動的な開始/停止

CICS プログラム・リスト・テーブル (PLT) を変更することによって、CICS リスナー・サポートを自動的に開始および停止することができます。

- スタートアップ (PLTPI)

  CICS リスナー・サポートを自動的に開始するには、PLTPI 内の以下の項目を DFHDELIM 項目の後に置きます。

  ```
  DFHPLT       TYPE=ENTRY,PROGRAM=EZACIC20
  ```

- シャットダウン (PLTSD)

  CICS リスナー・サポートを自動的にシャットダウンするには、PLTSD 内の以下の項目を DFHDELIM 項目の前に置きます。

  ```
  DFHPLT       TYPE=ENTRY,PROGRAM=EZACIC20
  ```

CICS リスナー・サポートの手動での開始/停止

EZAO トランザクションを使用することによって、CICS リスナー・サポートを手動で開始することができます。この操作トランザクションには、次の 4 つの機能があります。

CICS リスナー・サポートのスタートアップ
: CICS アドレス・スペース内で CICS リスナー・サポートを開始し、即時開始と指示されているすべてのリスナーを開始します。

  注: CICS リスナー・サポートを開始する CICS 上で EZAO トランザクションが実行していなければなりません。別の CICS から CICS リスナー・サポートを開始することはできません。

CICS リスナー・サポートのシャットダウン
: CICS アドレス・スペース内でこのインターフェースを停止します。

リスナーのスタートアップ
: CICS アドレス・スペース内でリスナーを開始します。

リスナーのシャットダウン
: CICS アドレス・スペース内でリスナーを停止します。

EZAO を入力すると、次の画面が表示されます。

```
EZAO                                                    APPLID=DBDCCICS
 ENTER ONE OF THE FOLLOWING
STArt
STOp









PF 1 HELP      3 END           6 CRSR          9 MSG           12 CNCL
```

START 機能

START 機能は、CICS リスナー・サポート、または単一のリスナーを開始します。

CICS リスナー・サポートが開始されると、即時開始とマークされたすべてのリスナーも開始されます。前の画面で STA を入力するか、空白画面で EZAO STA を入力すると、次の画面が表示されます。

```
EZAO START                                                  APPLID=DBDCCICS
 ENTER ONE OF THE FOLLOWING

 CICS         ===> ...                Enter Yes|No
 LIStener     ===> ...                Enter Yes|No









 PF 1 HELP       3 END            6 CRSR            9 MSG            12 CNCL
```

CICS リスナー・サポートの開始

EZAO START CICS を入力すると、次の画面が表示されます。

```
EZAO START CICS                                             APPLID=DBDCCICS

 CICS         ===> APPLID             APPLID of CICS









 RESULT MESSAGE APPEARS HERE

 PF 1 HELP       3 END            6 CRSR            9 MSG            12 CNCL
```

リスナーの開始

EZAO START LISTENER を入力すると、次の画面が表示されます。

```
EZAO START LISTENER                                          APPLID=DBDCCICS
 CICS         ===> APPLID             APPLID of CICS
 NAME         ===>                    Enter Name of Listener









 PF             3 END                              9 MSG            12 CNCL
```

リスナー名を入力すると、そのリスナーが開始されます。次の画面が表示され、結果がメッセージ領域に示されます。

```
EZAO START LISTENER                                          APPLID=DBDCCICS

 CICS         ===> APPLID             APPLID of CICS system
 NAME         ===> XXXX               Transaction Name of Listener









 RESULT MESSAGE APPEARS HERE

 PF             3 END                              9 MSG            12 CNCL
```

STOP 機能

STOP 機能は、CICS リスナー・サポート、またはインターフェース内の単一のリスナーを停止するのに使用します。

CICS リスナー・サポートが停止される場合、CICS リスナー・サポートの停止前に、すべてのリスナーが停止されます。前の画面で STO を入力するか、空白画面で EZAO STO を入力すると、次の画面が表示されます。

```
EZAO STOP                                                    APPLID=DBDCCICS
 ENTER ONE OF THE FOLLOWING

 CICS         ===> ...                Enter Yes|No
 LIStener     ===> ...                Enter Yes|No









 PF 1 HELP       3 END            6 CRSR            9 MSG            12 CNCL
```

CICS リスナー・サポートの停止

EZAO STOP CICS を指定すると、次の画面が表示されます。

```
EZAO STOP CICS                                               APPLID=DBDCCICS

 CICS         ===> ...                APPLID of CICS
 IMMEDIATE    ===> ...                Enter Yes|No









 PF 1 HELP       3 END            6 CRSR            9 MSG            12 CNCL
```

CICS リスナー・サポートの停止に関して、次の 2 つのオプションがあります。

IMMEDIATE=NO

このオプションは、CICS リスナー・サポート を制御して終了できるので、ほとんどの場合に使用する必要があります。この API を使用するアプリケーションに対して次のように作用します。

- リスナー・トランザクション (EZAL) は、最大 3 分間待った後、他にアクティブまたは中断状態のソケット・アプリケーションがなければ、静止します。

- アクティブまたは中断状態のソケット・アプリケーションがある場合、リスナーは、それらのアプリケーションが処理を続行することを許容します。それらのタスクがすべて終了したら、リスナーは終了します。
- 新規リスナーを開始することはできません。

IMMEDIATE=YES

このオプションは、通常は起こらない状況のために確保されているもので、CICS リスナー・サポートを急に終了させます。この API を使用するアプリケーションに対して次のように作用します。

- マスター・サーバー (リスナー) EZAL を強制的にパージします。

オプションを選択すると、停止が試みられます。画面が再表示され、結果がメッセージ行に示されます。

リスナーの停止

STOP LISTENER を指定すると、次の画面が表示されます。

```
EZAO STOP LISTENER                                          APPLID=DBDCCICS
 CICS         ===> DBDCCICS           APPLID of this CICS
 LIStener     ===> ........           Enter Name of Listener
 PF 1 HELP       3 END            6 CRSR            9 MSG            12 CNCL
```

リスナー名を入力すると、そのリスナーが停止されます。画面が再表示され、結果がメッセージ行に示されます。

プログラム・リンクを使用した CICS リスナー・サポートの開始/停止

プログラム EZACIC20 への EXEC CICS LINK の発行によって、CICS TCP/IP リスナー・サポートを開始または停止できます。LINK を行う側のプログラムには、以下のステップを組み込む必要があります。

1. EZACIC20 用の COMMAREA を定義する。これを行うには、DFHEISTG 定義内に以下の命令を組み込みます。

   ```
   EZACICA AREA=P20,TYPE=CSECT
   ```

   この領域の長さは P20PARML と等価であり、構造体の名前は P20PARMS です。

2. COMMAREA 内の値を次のように初期設定する。

P20TYPE

I 初期設定

T 即時終了

D 据え置き終了

P20OBJ

C CICS リスナー・サポート

L リスナー

P20LIST

これがリスナーの初期設定/終了である場合、リスナーの名前。

3. プログラム EZACIC20 に EXEC CICS LINK を発行する。EZACIC20 は、機能が完了するまで戻りません。
4. P20RET フィールドで EZACIC20 からの応答をチェックする。

注**:** EZACIC20 は、以下の異常終了コードを発行することがあります。

- CICS リスナー・サポートがスタートアップまたは終了されず、COMMAREA が用意されていなかった場合、E20L が発行されます。
- CICS リスナー・サポートがアクティブでない場合、E20T が発行されます。

第 21 章 ユーザー独自のリスナーの作成

基本的な要件

CICS リスナー・サポートでは、255 個までのリスナーをサポートする構造体が提供されています。IBM 提供のリスナーまたはユーザー作成リスナーのどちらか、または両方を組み合わせた、複数のリスナーを使用できます。あるいは、リスナーなしでの実行を選択することもできます。

それぞれの (IBM 提供の、またはユーザー作成の) リスナーには、インターフェースがリスナーを (特に初期設定および終了中に) 正しく管理できるようにするための一定の基本的な要件があります。それらの要件は、次のとおりです。

- 各リスナー・インスタンスには、同じリスナーの複数のコピーを実行している場合でも、それぞれ固有のトランザクション名がなければなりません。
- 各リスナーごとに、CICS リスナー構成データ・セット内にそれぞれ 1 つの項目がなければなりません。もし項目がないと、リスナーの自動開始を使用しない場合でも、終了処理が正しく行われず、CICS が正常にシャットダウンを完了できないことがあります。

IBM 提供のリスナーについては、 591 ページの『リスナー』を参照してください。

前提条件

インストール環境によっては、カスタマイズされたユーザー作成リスナーが必要です。ユーザー独自のリスナーを作成するには、以下の前提条件があります。

1. IBM 提供リスナーではサポートされない機能で、必要な機能が何であるかを判断します。その機能は、リスナーの一部なのか、サーバーの一部なのかを判別する必要もあります。
2. CICS アセンブラー環境の知識が必要です。
3. マルチスレッド化アプリケーションの知識が必要です。良好なパフォーマンスを得るためには、1 つのリスナーが複数の機能を同時に実行できる必要があります。
4. CICS リスナー・インターフェースの知識が必要です。

IBM の環境サポートの使用

ユーザー作成リスナーは、IBM 提供リスナーによって使用され、提供される環境サポートを使用できます。このサポートを利用する場合、ユーザー作成リスナーは、上記の要件に加えて、次のことを行わなければなりません (参照されるストレージ域についての詳しい説明は、 573 ページの『第 22 章 外部データ構造体』にあります)。

- ユーザー作成リスナーは、アセンブラーで作成する必要があります。

- リスナー・トランザクションおよびプログラムの RDO 定義は、トランザクション/プログラムの名前を除いて、IBM 提供リスナーのものと同じである必要があります。
- プログラム内に、構成ファイル・レコード用の入力域を定義します。入力域は、MOVE モードを使用して構成ファイルを読み取る場合は、DFHEISTG 領域に次の項目を作成することによって定義できます。

```
EZACICA AREA=CONFIG,TYPE=CSECT
```

  LOCATE モードを使用して構成ファイルを読み取る場合は、入力域用の DSECT を次のように定義できます。

```
EZACICA AREA=CONFIG,TYPE=DSECT
```

  どちらの場合も、領域の長さは、EQUATE ラベル CFGLEN で表されます。領域/DSECT の名前は CFG0000 です。
- CICS TCP/IP リスナー・サポートは、LE ランタイム環境を必要としません。従って、リスナー・プログラムを、アセンブラーのメインルーチンとして作成できます。
- プログラム内に、グローバル作業域 (Global Work Area (GWA)) をマッピングするための DSECT を定義します。これは、以下のマクロを発行することによって行われます。

```
EZACICA AREA=GWA,TYPE=DSECT
```

  この DSECT の名前は GWA0000 です。
- プログラム内に、リスナー制御域 (Listener Control Area (LCA)) をマッピングするための DSECT を定義します。これは、以下のマクロを発行することによって行われます。

```
EZACICA AREA=LCA,TYPE=DSECT
```

  この DSECT の名前は LCA0000 です。
- グローバル作業域 (GWA) のアドレスを取得します。これは、以下の CICS コマンドを使用して行うことができます。

```
EXEC  CICS EXTRACT EXIT PROGRAM(EZACIC01) GASET(ptr) GALEN(len)
```

  ここで、*ptr* はレジスターであり、*len* はハーフワード 2 進数の変数です。*ptr* に GWA のアドレスが戻され、*len* に GWA の長さが戻されます。
- リスナーの初期設定中に、構成ファイルを読み取ります。構成ファイルは、CICS 構成ファイル内で EZACONF として識別されます。ユーザー作成リスナーのレコード・キーは、次のとおりです。
  - APPLID

    8 バイトの文字フィールドであり、この CICS の APPLID 値に設定されます。この値は、GWA 内の GWACAPPL フィールドから取得するか、以下の CICS コマンドを使用して取得できます。

```
EXEC  CICS ASSIGN APPLID(applid)
```

    ここで、*applid* は 8 バイトの文字フィールドです。
  - レコード・タイプ

1 バイトの文字フィールドであり、レコード・タイプに設定されます。値は 'L' でなければなりません。

– 予約フィールド

3 バイトの 16 進フィールドであり、2 進ゼロに設定されます。

– トランザクション

4 バイトの文字フィールドであり、このリスナーのトランザクション名を含みます。これは、実行インターフェース・ブロックの EIBTRNID フィールドから取得できます。

構成レコードは、構成マクロまたは EZAC トランザクションのいずれかを介して入力された情報を提供します。ユーザー作成リスナーは、この情報を選択的に使用できますが、ユーザー作成リスナーでは、ポート、バックログ、およびソケット数のデータを使用することを強くお勧めします。

共用ファイルの場合**:** ユーザー作成リスナーは、構成ファイルを読み取る場合、最初に EXEC CICS SET コマンドを発行して、ファイルを使用可能にし、オープンする必要があります。ファイル操作が完了したら、ユーザー作成リスナーは、EXEC CICS SET コマンドを発行して、ファイルを使用不可にし、クローズする必要があります。この操作を誤ると、特定の共用ファイル状況ではファイル・エラーが発生します。

- ユーザー作成リスナーは、それ自体のリスナー制御域 (LCA) を見つける必要があります。一連の LCA は連続的にストレージ内に配置されていて、先頭の領域が GWA 内の GWALCAAD フィールドによって示されます。正しい LCA は、LCATRAN フィールド内に、リスナーのトランザクション名を持っています。
- ユーザー作成リスナーは、LCA 内の LCASTAT フィールド、または GWA 内の GWATSTAT フィールドのいずれかをモニターして、シャットダウン状況を確認するべきです。どちらかのフィールドが即時シャットダウンが進行中であることを示している場合、ユーザー作成リスナーは、EXEC CICS RETURN を発行することによって終了して、インターフェースがソケット接続をクリーンアップできるようにするべきです。どちらかのフィールドが据え置き終了が進行中であることを示している場合、ユーザー作成リスナーは以下を実行するべきです。
  1. 保留中の接続があれば受け入れ、受動 (listen) ソケットをクローズする。
  2. トランザクション開始に関係するソケットの処理 (例えば、GIVESOCKET コマンドの処理) を完了する。処理が完了したら、それらのソケットをクローズする。
  3. すべてのソケットがクローズしたら、EXEC CICS RETURN を発行する。
- ユーザー作成リスナーは、ACCEPT や READ などの外部イベントに依存するブロックを暗黙に意味するソケット呼び出しを避けるべきです。それらの呼び出しの前には、LCA 内の ECB LCATECB を扱う、1 つの SELECTEX 呼び出しを置くべきです。この ECB は、即時終了が検出されたときにポストされ、その結果、SELECTEX は RETCODE 0、ERRNO 0 で完了します。プログラムは、このように SELECTEX が完了したときに ECB をチェックするべきです。なぜなら、これは、タイムアウトになったときの SELECTEX の完了の仕方と同じだからです。ECB のチェックは、第 3 バイトの X'80' (ポスト・ビット) を確認することで行えます。

この SELECTEX は、タイムアウト値を指定するべきです。そうすることによって、リスナーは、定期的に据え置き終了要求をチェックする方法を備えることができます。これがない場合、CICS リスナー据え置き終了または CICS 据え置き終了は完了できません。

第 22 章 外部データ構造体

外部データ構造体

ユーザーが使用できるデータ構造体は以下のとおりです。

構成データ・セットのレコード・フォーマット

DSECT/構造体名

CFG0000

構造体の長さ

CFGLEN

マクロ展開

```
EZACICA  AREA=CONFIG,TYPE=DSECT
EZACICA  AREA=CONFIG,TYPE=CSECT
```

表 *18.* 構成ファイルのフォーマット

フィールド名	フィールド・タイプ	説明	デフォルト値
CFHAPPL	8 バイト文字	このレコードが参照する CICS オブジェクトのアプリケーション ID	
CFHRTYPE	1 バイト文字	レコード・タイプ • C = CICS オブジェクト・レコード • L = リスナー・オブジェクト・レコード	
	3 バイト	予約済み	00
CICS レコード・フォーマット			
CFCTRAN	4 バイト 2 進	2 進ゼロ。	00000000
CFCTCPIP	8 バイト文字	TCP/IP のアドレス・スペース名	SOCKET00
CFCNOTSK	2 バイト 2 進	再使用可能タスクの数 (1)	20
CFCSTIME	2 バイト 2 進	キャッシュ最小リフレッシュ時間	15
CFCLTIME	2 バイト 2 進	キャッシュ最大リフレッシュ時間	30
CFCNORES	2 バイト 2 進	キャッシュ同時リゾルバー数	10
CFCDPRTY	2 バイト 2 進	サブタスクの限界優先順位 (1)	0
CFCENAME	4 バイト文字	TD エラー・キューの名前	EZAM

表 *18.* 構成ファイルのフォーマット *(*続き*)*

フィールド名	フィールド・タイプ	説明	デフォルト値
CFCOPT	1 バイト 2 進	CICS オプション 値　意味 **B'00000001'** タスクの標準メッセージを抑制 **B'00000010'** トレースを抑制 (1) **B'00000100'** OTE を使用 (1) **B'00001000'** PLT 即時シャットダウン **B'00010000'** アプリケーション・データを登録 (1)	
	1 バイト	予約済み	
CFCTERML	2 バイト 2 進	サブタスクの終了限界 (1)	
CFCTCBLM	4 バイト 2 進	OPEN API TCP 限界 (1)	
リスナーのレコード・フォーマット			
CFLTRAN	4 バイト文字	リスナーのトランザクション名	EZAL
CFLPORT	2 バイト 2 進	リスナーのポート番号	
CFLBKLOG	2 バイト 2 進	リスナーの BACKLOG 値	20
CFLNSOCK	2 バイト 2 進	リスナーが使用するソケットの数	50
CFLMMIN	2 バイト 2 進	入力メッセージの最小長	4
CFLLTIM	2 バイト 2 進	ACCEPT のタイムアウト値 (秒)	60
CFLRTIM	2 バイト 2 進	READ のタイムアウト値 (秒)	0
CFLGTIM	2 バイト 2 進	GIVESOCKET のタイムアウト値 (秒)	ACCEPT のタイムアウト値と同じ

表 *18.* 構成ファイルのフォーマット *(*続き*)*

フィールド名	フィールド・タイプ	説明	デフォルト値
CFLOPT	1 バイト 2 進	リスナーのオプション 値　意味 **B'00000001'** 即時スタートアップ **B'00000110'** メッセージ全体を変換 **B'00000010'** トランザクション・コードのみを変換 **B'00000100'** ユーザー・データのみを変換 **B'00010000'** データのみをピーク **B'00100000'** アウトバウンド・メッセージは EBCDIC **B'01000000'** これは拡張リスナーです **B'10000000'** リスナー AF は AF_INET6	B'00000111'
CFLSECXT	8 バイト文字	セキュリティー出口の名前	
	36 バイト	予約済み	
CFLCSTRN	4 バイト文字	子サーバー TRANID	
CFLCSSTT	2 バイト文字	子サーバーの始動タイプ	
CFLCSDLY	6 バイト文字	子サーバーの遅延間隔	
	1 バイト	予約済み	
CFLMSGLN	2 バイト 2 進	インバウンド・メッセージの長さ	
CFLOPT2	1 バイト 2 進	リスナーのオプション、バイト 2 (1)	
CFLUSRID	8 バイト文字	リスナー・ユーザー ID (1)	
	1 バイト	予約済み	
CFLRTYTIME	2 バイト 2 進	スタック接続の再試行時間	

(1) z/VSE では使用されません。

グローバル作業域

DSECT/構造体名
GWA0000

構造体の長さ
GWALENTH (固定域の長さ)

マクロ展開

```
EZACICA  AREA=GWA,TYPE=DSECT
```

EZACICA AREA=GWA,TYPE=CSECT

表 *19.* グローバル作業域のフォーマット

フィールド名	フィールド・タイプ	説明	デフォルト値
グローバル作業域の始まりを示す目印			
GWACMDSC	8 バイト文字	識別子	EZACICGW
始動モジュールからの継承の始まり			
GWACMNAM	8 バイト文字	始動モジュール名	EZACIC21
	1 バイト文字	区切り文字	
GWACMVER	3 バイト文字	スタートアップ・サービス・レベル	
	1 バイト文字	区切り文字	
GWACMREL	11 バイト文字	始動モジュール日付または APAR	
	6 バイト文字	予約済み	
始動モジュールからの継承の終わり			
タスク関連ユーザー出口からの継承の始まり			
GWATRNAM	8 バイト文字	タスク関連ユーザー出口モジュール名	EZACIC01
GWATRVER	2 バイト文字	タスク関連ユーザー出口のバージョン番号 (1)	
GWATRREL	2 バイト文字	タスク関連ユーザー出口のリリース番号 (1)	
GWATRMOD	2 バイト文字	タスク関連ユーザー出口のモディフィケーション番号 (1)	
GWATRDAT	8 バイト文字	タスク関連ユーザー出口のアセンブルされた日付 (1)	
GWATRTIM	8 バイト文字	タスク関連ユーザー出口のアセンブルされた時間 (1)	
タスク関連ユーザー出口からの継承の終わり			
IBM リスナーからの継承の始まり			
GWAMSNAM	8 バイト文字	IBM リスナー・モジュール名	EZACIC02
GWAMSVER	2 バイト文字	IBM リスナーのバージョン番号	
GWAMSREL	2 バイト文字	IBM リスナーのリリース番号	
GWAMSMOD	2 バイト文字	IBM リスナーのモディフィケーション番号	
GWAMSDAT	8 バイト文字	IBM リスナーのアセンブルされた日付	
GWAMSTIM	8 バイト文字	IBM リスナーのアセンブルされた時間	
IBM リスナーからの継承の終わり			
生成済みサブタスクからの継承の始まり			
GWASTNAM	8 バイト文字	生成済みサブタスクのモジュール名 (1)	
GWASTVER	2 バイト文字	生成済みサブタスクのバージョン番号 (1)	
GWASTREL	2 バイト文字	生成済みサブタスクのリリース番号 (1)	
GWASTMOD	2 バイト文字	生成済みサブタスクのモディフィケーション番号 (1)	
GWASTDAT	8 バイト文字	生成済みサブタスクのアセンブルされた日付 (1)	
GWASTTIM	8 バイト文字	生成済みサブタスクのアセンブルされた時間 (1)	
生成済みサブタスクからの継承の終わり			
GWACMIBM	154 バイト文字	著作権文	
GWAENQTID	4 バイト文字	GWA をエンキューするトランザクション ID (1)	
GWAENQTNO	4 バイト 2 進	GWAENQTID のタスク番号 (1)	
	2 バイト	予約済み	

表 19. グローバル作業域のフォーマット (続き)

フィールド名	フィールド・タイプ	説明	デフォルト値
GWAC01TR	4 バイト 2 進	EZACIC01 の TRU00000 アドレス (1)	
GWANCHND	4 バイト 2 進	LCA チェーンの番号項目 (1)	
GWATCBLM	4 バイト 2 進	OTE の TCB 制限 (1)	
GWAOATNO	4 バイト 2 進	アクティブな OPEN API TCB 数 (1)	
GWAOATQD	4 バイト 2 進	TCBLIM キュー項目数 (1)	
GWAOATHW	4 バイト 2 進	延期タスクの最高水準点 (1)	
GWAOPT3	1 バイト	未使用オプション・フラグ	
GWAOPT2	1 バイト	未使用オプション・フラグ	
GWACTL	1 バイト	TCBLIM フラグ (1)	
GWAOPT	1 バイト	値 意味 **B'00000001'** タスクの標準メッセージを抑制 **B'00000010'** トレースを抑制 (1) **B'00000100'** OTE を使用 (1) **B'00001000'** PLT 即時シャットダウン **B'00010000'** アプリケーション・データを登録 (1)	
GWASTGPR	4 バイト 2 進	CICS ストレージ・プロテクター標識	
GWAUSCNT	4 バイト 2 進	この GWA の使用回数	
GWABKWRD	4 バイト 2 進	タスク・チェーン・アンカー後方アドレス (1)	
GWAFOWRD	4 バイト 2 進	タスク・チェーン・アンカー前方アドレス (1)	
GWACAPPL	8 バイト文字	CICS システムの VTAM APPLID	
GWATRUEN	8 バイト文字	タスク関連ユーザー出口ロード・モジュールの名前	
GWASTSKN	8 バイト文字	生成済みサブタスクのロード・モジュールの名前 (1)	
GWATCPID	8 バイト文字	TCP/IP アドレス・スペース名	SOCKET00
GWALCAAD	4 バイト 2 進	先頭のリスナー制御域のアドレス	
GWA03PSA	4 バイト 2 進	EZASOH03 ロード・モジュールのアドレス (1)	
GWANTASK	2 バイト 2 進	再使用可能タスクの数 (1)	
GWANLIST	2 バイト 2 進	リスナーの数	
GWATSTAT	1 バイト文字	タスク関連ユーザー出口状況 値 意味 **E** TRUE は使用可能 **I** 即時シャットダウン要求済み/実行中 **Q** 静止シャットダウン要求済み/実行中 **T** TRUE はシャットダウンを要求中	

表 19. グローバル作業域のフォーマット (続き)

フィールド名	フィールド・タイプ	説明	デフォルト値
GWARSHUT	1 バイト文字	EZAO シャットダウン要求標識 **値** 意味 **I** 即時シャットダウン要求済み **Q** 静止シャットダウン要求済み	
GWACSTART	1 バイト 2 進	CICS 実行状況 (1)	
GWAVOSYS	1 バイト 2 進	MVS バージョン (1)	
GWAOPREL	2 バイト 2 進	MVS リリース (1)	
GWACIVER	2 バイト文字	CICS バージョン番号 (1)	
GWACIREL	1 バイト文字	CICS リリース番号 (1)	
GWACIMOD	1 バイト文字	CICS モディフィケーション番号 (1)	
GWATOKEN	8 バイト文字	OS/390 登録/登録解除のトークン (1)	
GWAMSGMD	8 バイト文字	メッセージ・モジュールの名前	EZACICMx
GWATDMSG	4 バイト文字	メッセージの TD キュー	
GWACSTIM	2 バイト 2 進	キャッシュ・リフレッシュ短時間	
GWACLTIM	2 バイト 2 進	キャッシュ・リフレッシュ長時間	
GWACNRES	2 バイト 2 進	リゾルバーの最大数	
GWAC01EP	4 バイト 2 進	EZACIC01 のエントリー・ポイント・アドレス (1)	
GWAC02EP	4 バイト 2 進	EZACIC02 のエントリー・ポイント・アドレス (1)	
GWALCACH	4 バイト 2 進	最初のチェーン LCA のアドレス (1)	
GWATERML	2 バイト 2 進	サブタスクの終了限界 (1)	
GWALCACT	2 バイト 2 進	アクティブ LCA の数	
GWAMONST	4 バイト 2 進	TRUE EMP の使用不可状況 (1)	
GWAMONSTL	4 バイト 2 進	リスナー EMP の使用不可状況 (1)	
GWAMONCT	16 バイト 2 進	システム終了 EMP のカウンター (1)	
GWA の固定部分の終わり			

(1) z/VSE 内では使用されません

EZACIC20 のパラメーター・リスト (COMMAREA)

DSECT/構造体名
P20PARMS

構造体の長さ
P20PARML

マクロ展開

```
EZACICA  AREA=P20,TYPE=DSECT
EZACICA  AREA=P20,TYPE=CSECT
```

表 *20. EZACIC20* 用 *COMMAREA* のフォーマット

フィールド名	フィールド・タイプ	説明	デフォルト値
P20TYPE	1 バイト文字	関数の種類 値　意味 **I**　初期設定 **T**　即時終了 **D**　据え置き終了 **Q**　QUERY PLTSDI	
P20OBJ	1 バイト文字	初期設定/終了オブジェクト 値　意味 **C**　CICS 初期設定/終了 **L**　リスナーの初期設定/終了	
P20LIST	4 バイト文字	リスナーのトランザクション名	
P20RET	1 バイト 2 進	返されるフラグ 値　意味 **B'00000000'** エラーは検出されませんでした **B'00000001'** CICS ソケットの I/F 初期設定でのエラー **B'00000010'** リスナー初期設定のエラー **B'00000100'** CICS ソケットの I/F 終了でのエラー **B'00001000'** リスナー終了でのエラー **B'00010000'** COMMAREA コンテンツでのエラー **B'00100000'** CICS/TS for VSE でのエラー	

リスナー制御域 (LCA)

DSECT/構造体名
LCA0000

構造体の長さ
LCALEN

マクロ展開

```
EZACICA  AREA=LCA,TYPE=DSECT
EZACICA  AREA=LCA,TYPE=CSECT
```

表 21. リスナー制御域 (LCA)

フィールド名	フィールド・タイプ	説明	デフォルト値
LCATECB	フルワード 2 進	終了マネージャーによってポストされる ECB	
LCATRAN	4 バイト文字	このリスナーのトランザクション名	
LCASTAT	1 バイト 2 進	このリスナーの状況、バイト 1 値　意味 **B'00000000'** リスナーは作動していない **B'00000001'** リスナーは初期化中 **B'00000010'** リスナーは SELECT 中 **B'00000100'** リスナーは処理中 **B'00001000'** リスナーに初期化エラーが発生 **B'00010000'** 即時終了が実行中 **B'00100000'** 据え置き終了が実行中 **B'01000000'** リスナーがアクティブ **B'10000000'** リスナーは CICS 遅延再試行	
LCASTAT2	1 バイト 2 進	このリスナーの状況、バイト 2 値　意味 **B'00000001'** リスナーは現在 TCP/IP に接続可能 **B'00000010'** アプリケーション・データを登録 (1) **B'00000100'** LAPPLD は APPLDAT を継承 (1) **B'00100000'** これは標準リスナーです **B'01000000'** これは拡張リスナーです **B'10000000'** リスナー AF は AF_INET6	
	2 バイト	予約済み	
LCAIEAD	4 バイト 2 進	リスナーの TIE のアドレス	

第 23 章 CICS リスナーのプログラミング考慮事項

概説

このセクションでは、クライアント、子サーバー・プロセスが関連付けられている並行サーバー、および反復サーバーの各プログラムでの標準的な呼び出しシーケンスについて説明します。セクションの内容は次のとおりです。

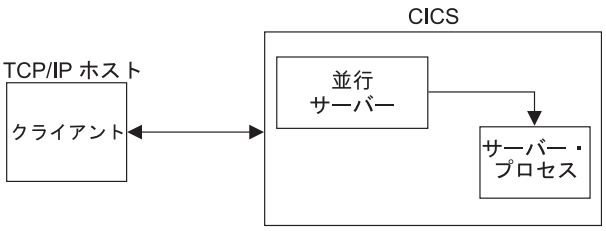
- CICS TCP/IP アプリケーションを作成するための、以下の 4 つのセットアップ。
  1. CICS TCP/IP 下で実行する、並行サーバー (提供されているリスナー・トランザクション) および子サーバー・プロセス
  2. 1 と同じだが、ユーザー作成の並行サーバーを使用する
  3. CICS TCP/IP 下で実行する反復サーバー
  4. CICS TCP/IP 下で実行するクライアント・アプリケーション
- ソケット・アドレス
- GETCLIENTID、GIVESOCKET、および TAKESOCKET コマンド
- リスナー・プログラム

CICS TCP/IP アプリケーションの作成

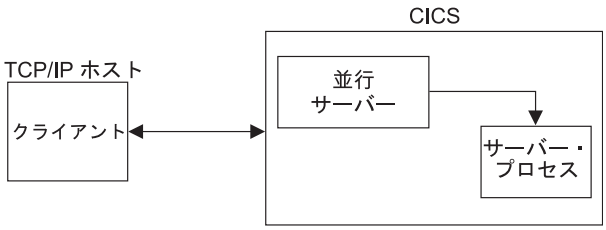
このセクションでは、CICS TCP/IP アプリケーションをクライアント/サーバー・システムのさまざまな部分で使用する場合の 4 つの TCP/IP セットアップについて詳しく説明します。

それらのセットアップは、以下のとおりです。

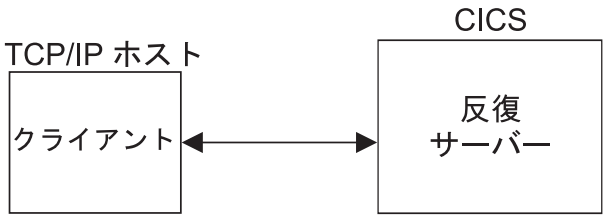
1. クライアント、リスナー、子サーバー・アプリケーションのセット。並行サーバーと子サーバー・プロセスが CICS TCP/IP 下で実行します。並行サーバーは、提供されたリスナー・トランザクションです。クライアントは、 OS/2 オペレーティング・システムまたはさまざまな UNIX オペレーティング・システムの 1 つ (AIX など) の下で TCP/IP を実行していてもかまいません。



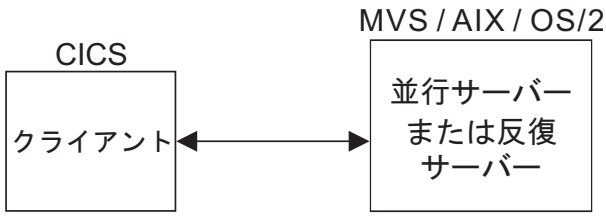
2. ユーザー独自の並行サーバーの作成。これは、IBM リスナーの代わりにユーザー作成の並行サーバーが使用されることを除いて、1 番目のセットアップと同じです。



3. 反復サーバー **CICS TCP/IP** アプリケーション。このセットアップは、一時点に 1 つのソケットを処理するように設計されています。



4. クライアント **CICS TCP/IP** アプリケーション。このセットアップでは、CICS アプリケーションはクライアントであり、サーバーはリモート TCP/IP プロセスです。



1. クライアント、リスナー、子サーバー・アプリケーションのセット

583 ページの図 53 に、このセットアップに関係する CICS コマンドおよびソケット呼び出しのシーケンスを示します。CICS コマンドの前には EXEC CICS が付いています。この図の中の番号が付いている他の項目はすべて CICS TCP/IP 呼び出しです。

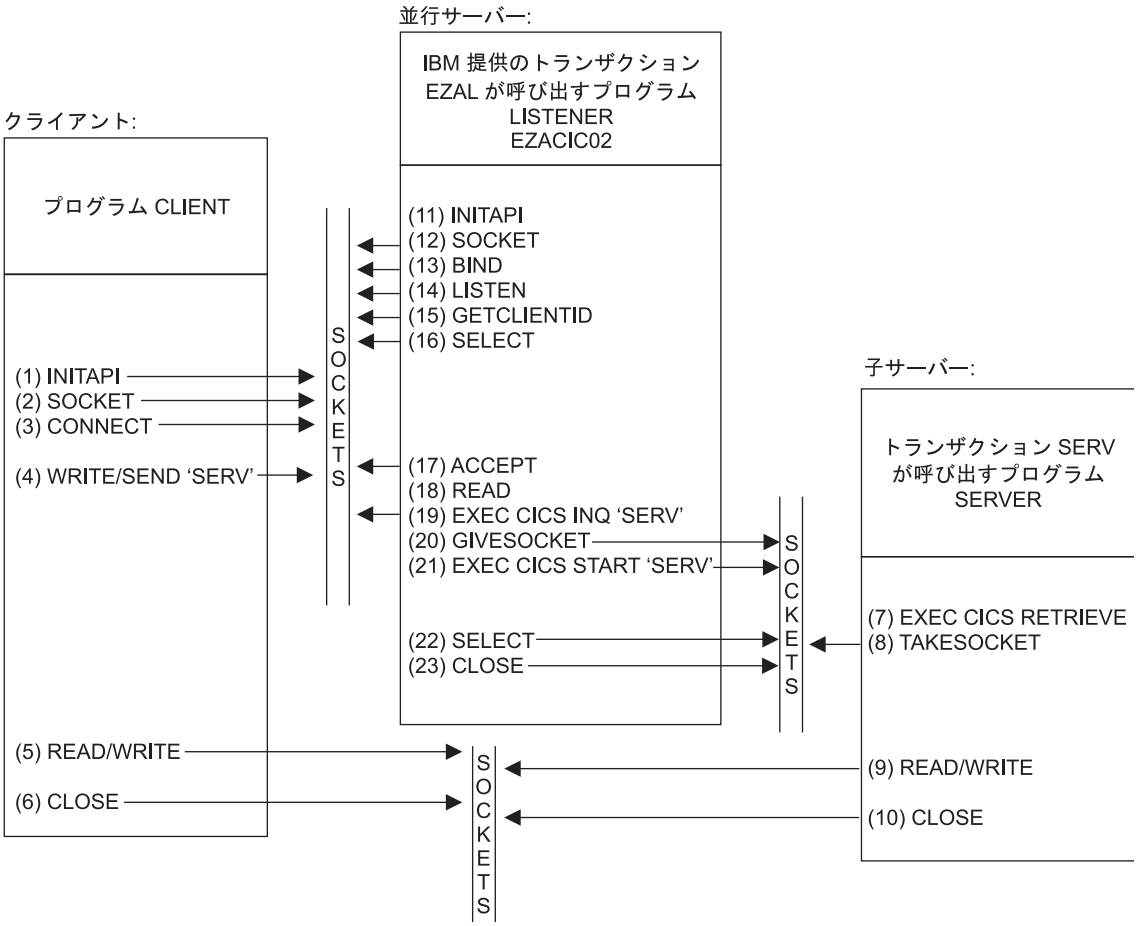


図 *53.* ソケット呼び出しのシーケンス

クライアントでの呼び出しシーケンス

表 22 は、図 53 に示された各呼び出しの機能説明です。

表 *22.* クライアント・アプリケーションでの呼び出し

(1) INITAPI	CICS アプリケーションが TCP/IP インターフェースに接続されます。MAX-SOC パラメーターを使用して、アプリケーションが使用する最大ソケット数を指定してください。

表 22. クライアント・アプリケーションでの呼び出し (続き)

(2) SOCKET	これによって、ソケットが取得されます。次の 3 つのパラメーターでソケットを定義してください。 • ドメイン、またはアドレス・ファミリー • ソケットのタイプ • プロトコル VSE TCP/IP の場合、ドメインに指定できるのは TCP/IP インターネット・ドメイン (COBOL の場合は 2、C の場合は AF_INET) のみです。タイプは、ストリーム・ソケット (COBOL の場合は 1、C の場合は SOCK_STREAM)、またはデータグラム・ソケット (COBOL の場合は 2、C の場合は SOCK_DGRAM) のいずれかです。プロトコルは、TCP または UDP のいずれかです。プロトコルに 0 を渡すと、デフォルトのプロトコルが選択されます。 成功した場合、SOCKET 呼び出しは、ソケット記述子 s を戻します。この値は常に短精度整数です。取得されたソケットは、ローカル・アドレスまたは宛先アドレスにはまだ接続されていないことに注意してください。
(3) CONNECT	クライアント・アプリケーションがこれを使用して、リモート・サーバーとの接続を確立します。この接続で使用されるローカル・ソケット s (上で取得されたもの) と、リモート・ソケットのアドレスおよびポート番号を定義する必要があります。ローカル・アドレスはシステムによって提供されます。CONNECT が成功して戻ると、ソケットは完全に定義され、TCP 接続 (ストリームの場合) または UDP 接続 (データグラムの場合) と関連付けられた状態になります。
(4) WRITE	これによって、最初のメッセージがリスナーに送信されます。メッセージには、先頭 4 バイトのデータとして CICS トランザクション・コードが含まれます。バッファー・アドレスおよび送信データの長さも指定する必要があります。
(5) READ/WRITE	これらの呼び出しは、サーバーが完了するまで、サーバーとの会話を続けます。
(6) CLOSE	これによって、指定されたソケットがクローズされ、接続は終了します。他のアプリケーションのためにソケットのリソースが解放されます。

リスナーでの呼び出しシーケンス

リスナー・トランザクション EZAL は CICS TCP/IP の一部として提供されています。これらの呼び出しは、CICS リスナーによって発行される呼び出しです。クライアントおよびサーバーでの呼び出しシーケンスは、このシーケンスと連携するように作成する必要があります。 585 ページの『2. ユーザー独自の並行サーバーの作成』に、これらの呼び出しの説明があり、 583 ページの図 53 に示されたリスナー呼び出しが説明されています。

子サーバーでの呼び出しシーケンス

585 ページの表 23 は、 583 ページの図 53 に示された各呼び出しの機能説明です。

表 23. サーバー・アプリケーションでの呼び出し

(7) EXEC CICS RETRIEVE	これは、並行サーバー・プログラム中の EXEC CICS START コマンドによって渡されるデータを取得します。このデータには、ソケット記述子、並行サーバーのクライアント ID が含まれ、オプションでクライアントからの追加データも含まれます。
(8) TAKESOCKET	これは、新しく作成されたソケットを並行サーバーから取得します。TAKESOCKET のパラメーターでは、取得するソケット記述子と、並行サーバーのクライアント ID を指定する必要があります。この情報は、EXEC CICS RETRIEVE コマンドで取得したものです。 **注:** TAKESOCKET は、最初の呼び出しである場合、デフォルト値を使用して暗黙に INITAPI を発行します。
(9) READ/WRITE	クライアントとの会話が、完了するまで続きます。
(10) CLOSE	これは、接続を終了し、完了時にソケットのリソースを解放します。

2. ユーザー独自の並行サーバーの作成

このセットアップは、全体では最初のシナリオと同じですが、リスナーで実行された機能の多くを並行サーバー・アプリケーションが実行します。

クライアントおよび子サーバー・アプリケーションは、同じ機能を実行します。

並行サーバーでの呼び出しシーケンス

表 24 は、 583 ページの図 53 に示された各ステップの機能の説明です。

表 24. 並行サーバー・アプリケーションでの呼び出し

(11) INITAPI	583 ページの表 22 に説明されているように、これはアプリケーションを TCP/IP に接続します。
(12) SOCKET	583 ページの表 22 に説明されているように、これはソケットを取得します。
(13) BIND	ソケットが取得された後、この呼び出しを使用して、並行サーバーは、それ自体を特定のアドレスで特定のポートにアタッチし、クライアントがその並行サーバーに接続できるようにします。引数として、ソケット記述子、ローカル・アドレス、およびポート番号が渡されます。 BIND 呼び出しが成功して戻ると、ソケットはローカル・アドレスのポートにバインド されていますが、リモート・アドレスには (まだ) バインドされていません。
(14) LISTEN	アドレスをソケットにバインドした後、並行サーバーは LISTEN 呼び出しを使用して、クライアントから接続を受け入れる準備ができていることを表します。LISTEN は、着信接続要求を並行サーバーが処理できるようになるまですべての着信接続要求が待ち行列に入れて保持される必要があることを、TCP/IP に知らせます。待ち行列の最大サイズは、この呼び出しの BACKLOG パラメーターで設定されます。

表 24. 並行サーバー・アプリケーションでの呼び出し (続き)

(15) GETCLIENTID	このコマンドは、TCP/IP が並行サーバーを認識するための識別子 (z/VSE パーティション名およびサブタスク名) を戻します。この情報は、EXEC CICS START 呼び出しで必要になります。
(16) SELECT	SELECT 呼び出しは、ソケットの集合のアクティビティーをモニターします。このケースでは、(LISTEN 呼び出しで作成された) 待ち行列を調べて接続があるかどうかをチェックするために使用されます。この呼び出しは、着信 CONNECT 呼び出しが受信されると戻りますが、そうでなければ、SELECT のパラメーターの 1 つで指定されたインターバルが経過した後にタイムアウトになります。
(17) ACCEPT	並行サーバーは、この呼び出しを使用して、待ち行列の最初にある着信接続要求を受け入れます。ACCEPT は、元のソケットと同じ特性を持つ新しいソケット記述子を作成します。元のソケットは、今までどおり接続要求を受け入れます。新しいソケットは、接続を開始したクライアントと関連付けられます。
(18) READ	これは、どのサービスが必要なのかを判別するため、クライアントからメッセージを 1 つ読み取ります。このメッセージには、最小限、サーバーの CICS トランザクション ID が含まれています。
(19) CICS INQ	これは、SERV トランザクションが CICS に定義されている (されていない場合は TRANSIDERR 例外条件が発生します) ことを確認し、定義されていれば、その状況が ENABLED であることを確認します。どちらかの確認が失敗すると、リスナーは SERV トランザクションの開始を試みません。
(20) GIVESOCKET	これは、ACCEPT 呼び出しで取得したソケットを、子サーバー・プログラムに使用可能にします。
(21) CICS START	これは、子サーバー・アプリケーション用の CICS トランザクションを開始し、GETCLIENTID で取得した並行サーバーの ID をサーバーに渡します。例えば、 593 ページの『リスナー出力フォーマット』では、パラメーター `LSTN-CLIENTID` がリスナーを定義しています。
(22) SELECT	TCP/IP アクティビティーをモニターするため、再度 SELECT 呼び出しが使用されます。今回の SELECT は、子サーバーが TAKESOCKET 呼び出しを発行したら戻ります。この SELECT は、ステップ 16 の SELECT 呼び出しと同じです。実行される機能を分かりやすく示すために 2 つの呼び出しとして記述されています。
(23) CLOSE	これは、子サーバーとの競合を避けるために、新しいソケットを解放します。

ソケットの引き渡し

ソケットは、同じタスク内のプログラム間で、ソケット記述子を受け渡しすることで引き渡すことができます。しかし、CICS タスク間でのソケットの引き渡しには、GIVESOCKET/TAKESOCKET をこの順序で呼び出すことが必要です。

3. 反復サーバー CICS TCP/IP アプリケーション

587 ページの図 54 に、クライアントと反復サーバーの単純なセットアップに関係するソケット呼び出しのシーケンスを示します。

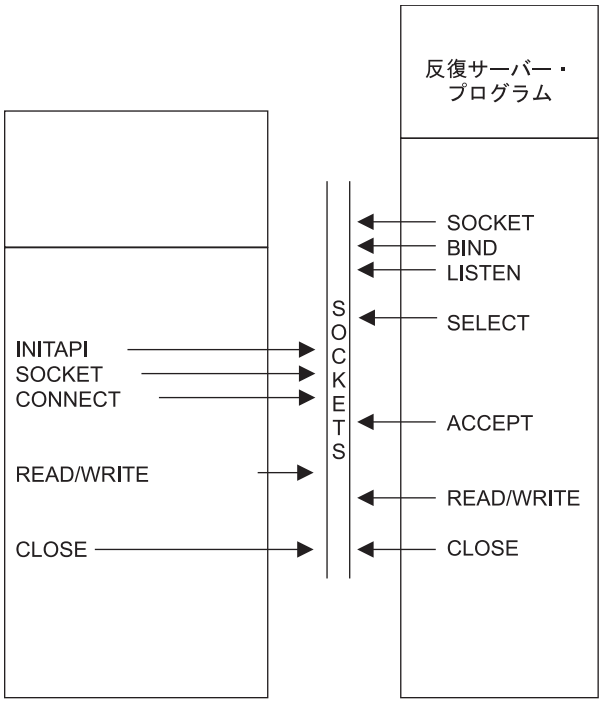


図 *54.* 反復サーバーでのソケット呼び出しシーケンス

反復サーバーを使用するこのセットアップは、並行サーバーを使用する上記のケースよりもずっと単純です。

反復サーバーのソケット使用法

反復サーバーに必要なことは、2 つのソケット記述子を取得することのみです。反復サーバーは、以下の呼び出しを行います。

1. 並行サーバーの場合と同様に、SOCKET、BIND、および LISTEN 呼び出しを使用して、サーバーが着信要求に対して準備ができていて、ソケット 0 で listen することを TCP/IP に知らせます。
2. 次に、SELECT 呼び出しが、接続要求が受信されると戻ります。これは ACCEPT 呼び出しの発行を促します。
3. ACCEPT 呼び出しは、新しいソケット (1) を取得します。ソケット 1 が使用されてトランザクションが処理されます。これが完了すると、ソケット 1 はクローズします。
4. SELECT 呼び出しに制御が戻り、次の接続要求を待ちます。

反復サーバーの欠点は、トランザクションの続いている間はブロックしたままになることです。

4. クライアント CICS TCP/IP アプリケーション

588 ページの図 55 に、CICS クライアントとリモート・サーバーのセットアップでの呼び出しシーケンスを示します。ここでの呼び出しも、上記の例に似た内容です。

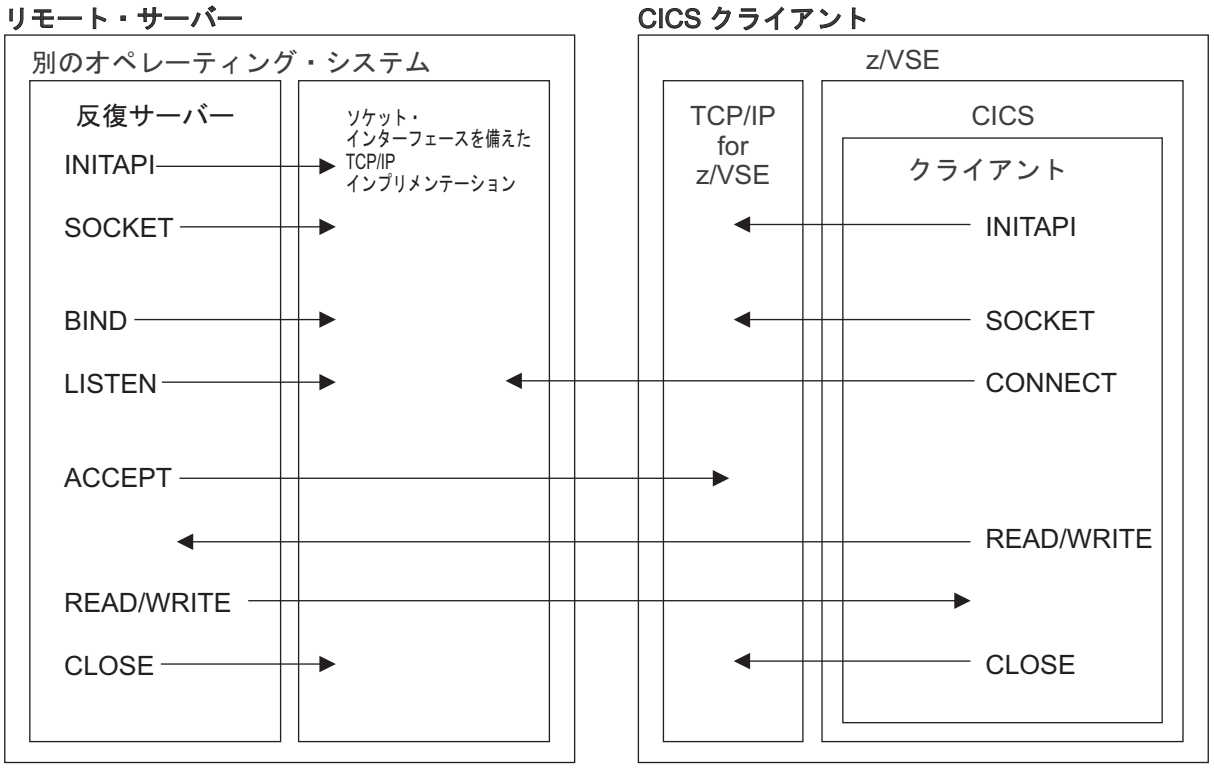


図 *55. CICS* クライアントとリモート反復サーバー間のソケット呼び出しシーケンス

図 55 に示すように、サーバーは、任意のプロセッサーに置くことができ、任意のオペレーティング・システムで実行できます。ただし、ソフトウェアとハードウェアの組み合わせが TCP/IP サーバーをサポートする構成である場合に限ります。

簡単にするために、図では 1 つの反復サーバーを示しています。並行サーバーがある場合は、リモート・プロセッサーにある子サーバーが必要であり、 583 ページの図 53 に示したモデルに従って呼び出しを調整することが必要になります。

CICS サーバーは、READ 呼び出しを発行して、必要な子サーバーの CICS トランザクション名が入っている、クライアントの最初のメッセージを読み取ります。サーバーが非 CICS システムにある場合、アプリケーション設計は、CICS クライアントからの最初のメッセージが、必要とされるサービスをどのように指示するのかを指定する必要があります (図 55 では、最初のメッセージは WRITE 呼び出しによって送信されます)。

サーバーが並行サーバーである場合は、子サーバーの名前によって指示されるのが一般的です。サーバーが図 55 に示されたように反復サーバーである場合で、すべてのクライアント呼び出しが同じサービスを必要とする場合、そのような指示は不要です。

ソケット・アドレス

ソケット・アドレスは、アドレス・ファミリーと、インターネット内のソケットのアドレスを指定することで定義されます。

VSE TCP/IP では、ソケットの IP アドレスとポート番号によってアドレスが指定されます。

アドレス・ファミリー (ドメイン)

VSE TCP/IP がサポートするのは、TCP/IP アドレス・ファミリー (UNIX システムでの名前に従って、ドメインとも呼ばれます) のみです。これは、インターネット・ドメインであり、C では AF_INET と表されます。多くのソケット呼び出しでは、パラメーターの 1 つとしてドメインが定義される必要があります。

ソケット・アドレスは、ソケットの IP アドレスとソケットに割り振られたポート番号によって定義されます。

IP アドレス

TCP/IP インターネット上の各 TCP/IP アドレスには、IP アドレスが割り振られています。各アドレスは、それぞれ固有の 32 ビットであり、ホストのネットワークおよび特定ホストを定義します。複数のネットワークに接続されているホスト (いわゆるマルチホーム・ホスト) は、複数の IP アドレスを持つことができます。

ポート

1 つのホストが、同時にいくつかの TCP/IP 接続を保持できます。同じホスト上で TCP/IP を使用する 1 つ以上のアプリケーションは、ポート番号によって識別されます。ポート番号は、正しいアプリケーションにデータを渡すためにシステム・ソフトウェアが追加で使用する修飾子です。ポート番号は 16 ビットの整数です。いくつかの番号は、特定のアプリケーション用に予約済みであり、ウェルノウン・ポートと呼ばれます (例えば、TELNET 用の 23)。

アドレス構造体

IP アドレス・ファミリー中のソケット・アドレスは、アドレス・ファミリー、IP アドレス、ポート、文字列 (ゼロ) の 4 つのフィールドから構成されます。各フィールドは次のように設定されます。

- ファミリー・フィールドは、C の場合は AF_INET に、他の言語の場合は 2 に設定されます。
- ポート・フィールドは、アプリケーションによって使用されるポートであり、ネットワーク・バイト・オーダーで設定されます (ページ 『ネットワーク・バイト・オーダー』に説明があります)。
- アドレス・フィールドは、アプリケーションによって使用されるネットワーク・インターフェースの IP アドレスです。これもネットワーク・バイト・オーダーです。
- 文字配列フィールドは、常にすべてゼロに設定される必要があります。

ネットワーク・バイト・オーダー

ポートおよびアドレスは、ビッグ・エンディアン と呼ばれる TCP/IP ネットワーク・バイト・オーダー規約を使用して指定されます。

ビッグ・エンディアン・システムでは、最上位バイトが先頭になります。それとは対照的に、リトル・エンディアン・システムでは、最下位バイトが先頭にきます。

z/VSE は、ビッグ・エンディアン規則を使用します。これはネットワーク規約と同じであり、CICS TCP/IP アプリケーションが変換ルーチン (例えば、htonl、htons、ntohl、ntohs) を使用する必要がないためです。

注: ソケット・インターフェースは、アプリケーション・データ内のデータ・バイト・オーダーの違いを処理しません。ソケット・アプリケーションの作成者自身が、それらの違いに対処する必要があります。

GETCLIENTID、GIVESOCKET、および TAKESOCKET

CICS では、ソケット呼び出し GETCLIENTID、GIVESOCKET、および TAKESOCKET を EXEC CICS START および EXEC CICS RETRIEVE の各コマンドで使用して、ソケットを新しいプロセスで使用可能にします。図 56 にこれを示します。

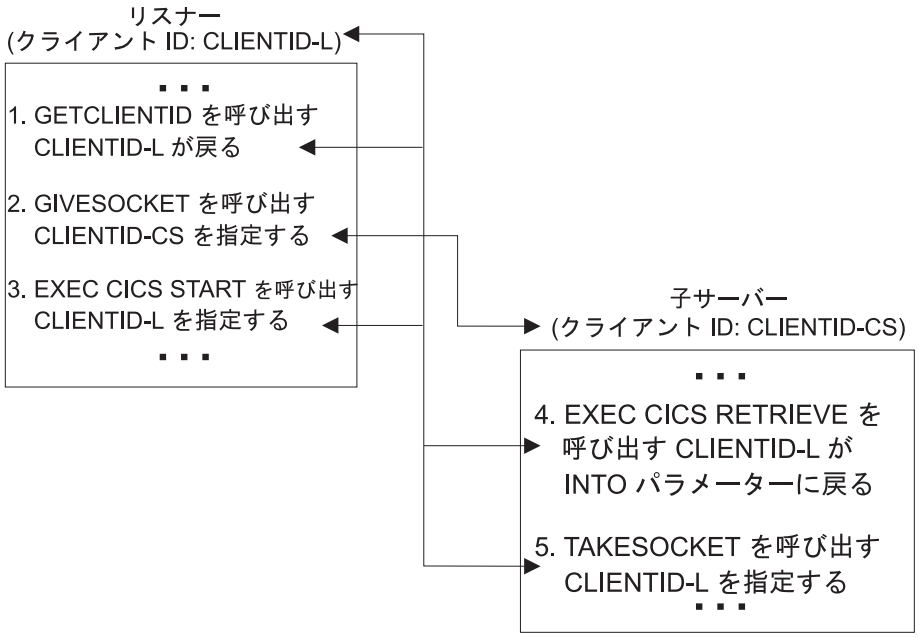


図 *56. CLIENTID* 情報の転送

図 56 は、リスナー・ソケットを子サーバー・プロセスに使用可能にするために使用される呼び出しを図示しています。以下のステップがあります。

1. リスナーは GETCLIENTID を呼び出します。これは、リスナー自体の CLIENTID (CLIENTID-L) を戻し、その中には、z/VSE パーティション名とリスナーのサブタスク ID が含まれています。リスナー・トランザクションは、ステップ 3 (591 ページ) 用に、それ自体の CLIENTID にアクセスする必要があります。
2. リスナーは、ソケット記述子と子サーバーの CLIENTID を指定して GIVESOCKET を呼び出します。

   リスナーと子サーバー・プロセスが、同じ CICS 領域にある場合 (従って、同じアドレス・スペースにある場合)、CLIENTID 内の z/VSE パーティション名識別子を空白に設定することができます。これは、リスナーのアドレス・スペースが子のアドレス・スペースでもあることを意味します。

リスナーと子サーバー・プロセスが、異なる CICS 領域にある場合は、新しいアドレス・スペースおよびサブタスクを入力してください。

CLIENTID 構造体には、提供されているリスナーによって、それ自体の z/VSE パーティション名が入れられ、サブタスク ID はブランクに設定されます。これは、リスナーのアドレス・スペース内の任意のタスクからの TAKESOCKET コマンドに対してソケットを使用可能にしますが、ソケット記述子を受け取ることができるのは子サーバーだけであり、ソケットの露出は最小限です。全体的な整合性のため、子サーバーのサブタスク ID を入力するべきです。

3. リスナーは EXEC CICS START を実行します。FROM パラメーターに、前の GETCLIENTID で取得した CLIENTID-L が指定されます。これによって、リスナーは、新規の子サーバーに、ステップ 5 でソケットを取得する元になる場所を知らせます。
4. 子サーバーは EXEC CICS RETRIEVE を実行します。INTO パラメーターに、CLIENTID-L が取得されます。
5. 子サーバーは、ソケットを取得する元のプロセスとして CLIENTID-L を指定して、TAKESOCKET を呼び出します。

リスナー

SNA 端末をベースにした CICS システムでは、CICS 端末管理モジュールが並行サーバーの機能を実行します。TCP/IP インターフェースは CICS 端末管理を使用しないため、CICS は、CICS アプリケーション・トランザクションの形式、つまりリスナーでこれらの機能を提供します。リスナーの CICS トランザクション ID は EZAL です。

リスナーは、以下の機能を実行します。

1. 適切な TCP/IP 呼び出しを発行することによって、構成ファイル内に指定されたポートで「listen」し、クライアントが発行する着信接続要求を待機します。
2. 着信接続要求が到着すると、リスナーはその要求を受け入れ、CICS 子サーバー・アプリケーション・プログラムに渡すための新規ソケットを取得します。
3. 標準リスナーは、新しい接続の最初のメッセージにある情報に基づいて CICS 子サーバー ・トランザクションを始動します。この情報のフォーマットは、592 ページの『リスナーの入力フォーマット (標準リスナーのみ)』で説明します。拡張リスナーは、TCP/IP CICS 構成ファイル EZACONF にある情報に基づいて CICS 子サーバー・トランザクションを始動します。
4. 子サーバー・トランザクションが新規ソケットを取得するのを待ち、その後でクローズ呼び出しを発行します。これが発生すると、受け取り側のアプリケーションは、ソケットの所有権を得たこと、リスナーはもうそのソケットには関与しないことを想定します。

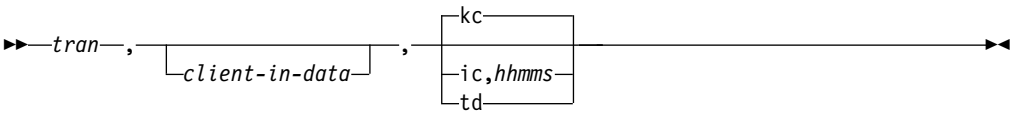
リスナー・プログラムは、このアクティビティーの一部が並行して行われるように作成されています。例えば、プログラムは、新規サーバーが新規ソケットを受け入れるのを待っている間に、次の着信接続がないか listen します。プログラムでは、49 個の子サーバーを同時に開始するという処理が可能です。開始処理は、リスナーが接続を受け入れたときに始まり、リスナーが子サーバーに与えたソケットをクローズしたときに終わります。

リスナーの入力フォーマット (標準リスナーのみ)

リスナーに、クライアントから最初の送信を行う場合、以下の入力フォーマットを使用する必要があります。クライアントは、応答を待機し、その後で後続の送信を行います。入力は、大文字でも小文字でもかまいません。コンマは必須です。

注: リスナーは、リスナーの初期メッセージで区切り文字として使われるコンマと、client-in-data フォーマットの一部としてのコンマとの識別ができません。そのため、client-in-data フォーマットはコンマを含むことはできません。ASCII データの x'2C' や EBCDIC データの '6B' のようなテキストでは、単一引用符がコンマとして解釈される場合があります。

フォーマット



パラメーター

tran
リスナーが開始する CICS トランザクション ID (大文字) です。このフィールドは、1 から 4 文字です。

client-in-data
このオプションのパラメーターは、オプションのセキュリティー出口 (594 ページの『リスナー用のユーザー独自のセキュリティー・リンク・モジュールの作成』 を参照してください) またはサーバー・トランザクションが使用するアプリケーション・データを指定します。このフィールドの最大長は 40 バイトの文字 (35 バイト、1 バイトの充てん文字、および 4 バイトの始動タイプ) です。

kc|ic|td
このオプション・パラメーターでは、CICS 始動制御の場合は kc、CICS インターバル制御の場合は ic、また CICS 一時データの場合は td という、始動タイプを指定します。タイプには大/小文字の区別はありません。このフィールドをブランクにすると、CICS タスク制御 (kc) で即時に始動します。kc は、子サーバー・タスクが遅延間隔なしで **EXEC CICS START** を使用して始動することを示します。これは、**IC,000000** を指定するのと同じです。

hhmmss
トランザクションがインターバル制御 (ic) を使用して始動する場合、このパラメーターは必須です。インターバル時間の時、分、秒を指定します。6 桁すべてが指定されなければなりません。

注: TD は、この時間フィールドを無視します。

例

以下に、クライアント入力とその結果のリスナー処理の例を示します。参照先のデータ・フィールドは、 593 ページの『リスナー出力フォーマット』 にあります。パラメーターはコンマで区切ります。

例	リスナーの応答
TRN1,userdataishere	タスク制御を使用して CICS トランザクション TRN1 を開始し、データ userdataishere を CLIENT-IN-DATA フィールドに渡します。
TRN2,,IC,000003	インターバル制御を使用して、ユーザー・データなしで CICS トランザクション TRN2 を開始します。リスナーからのスタートアップ要求と CICS でのトランザクション・スタートアップとの間には、3 秒の遅延があります。
TRN3,userdataishere,TD	TRN3 という名前の一時データ待ち行列に、TCPSOCKET-PARM 構造体 (『リスナー出力フォーマット』 に説明があります) で記述される形式でメッセージを書き込みます。userdataishere に入っているデータが CLIENT-IN-DATA フィールドに渡されます。このキューは、トリガー・レベルが 1 に設定された区画内キューでなければなりません。この設定では、トランザクション TRN3 がまだアクティブでなければ開始されます。このトランザクションは、一時データ待ち行列を読み取り、待ち行列が空になるまで要求を処理するようにコーディングされている必要があります。 この仕組みは、各サーバー要求に対して別個に CICS トランザクションを開始するとオーバーヘッドがパフォーマンスに影響を与えるような、頻繁に使用されるサーバー・トランザクションのために用意されています。
TRN3,,TD	この場合、データは一時データ待ち行列 TRN3 に入れられ、上記の TRN3 例で説明されているように、CICS トランザクション TRN3 の開始または続きの処理が行われます。渡されるユーザー・データはありません。
TRN4	これは、タスク制御を使用して CICS トランザクション TRN4 を開始します。この新規トランザクションに渡されるユーザー・データはありません。

リスナー出力フォーマット

594 ページの表 25 に、子サーバーに渡される、リスナー出力データ域のフォーマットを示します。この出力データの全体域の長さは 96 バイトです。リスナー・プログラムは、以下の COBOL 定義を使用します。

```
01  TCPSOCKET-PARM.
    05  GIVE-TAKE-SOCKET    PIC 9(8) COMP.
    05  LSTN-CLIENTID.
        15 LSTN-CID-DOMAIN  PIC 9(8) COMP.
        15 LSTN-CID-NAME    PIC X(8)
        15 LSTN-CID-TASK    PIC X(8)
        15 LSTN-CID-RSVD    PIC X(20)
    05  CLIENT-IN-DATA      PIC X(35).
    05  FILLER              PIC X(1).
    05  SOCKADDR-IN-PARM.
        15 SIN-FAMILY       PIC 9(4) COMP.
        15 SIN-PORT         PIC 9(4) COMP.
        15 SIN-ADDRESS      PIC 9(8) COMP.
        15 SIN-ZERO         PIC X(8).
```

表 25. リスナー出力フォーマット - *IPv4* プロトコルの使用

説明	フォーマット	値
ソケット記述子	フルワード 2 進	子サーバーの TAKESOCKET コマンドで使用されるソケット記述子
リスナー・クライアント ID	40 バイト	リスナーのクライアント ID
データ域	35 バイトの文字と 1 バイトの充てん文字	クライアントから受け取った、リスナーの client-in-data 入力
ソケット・アドレス	残りの 4 つのフィールドを含む構造体	各フィールドを参照
TCP/IP アドレス・ファミリー	ハーフワード 2 進	AF-INET を示す 2
ポート記述子	ハーフワード 2 進	ソケットにバインドされたポートの記述子 (構成ファイルにある、リスナーのポート番号)。
32 ビット IP アドレス	フルワード 2 進	ネットワーク・バイト・オーダーで表された、ソケットのホスト・マシンの IP アドレス
未使用	ダブルワード	2 進ゼロ。

表 26. *EZA* リスナーから子サーバーへの出力 - 拡張モード - *IPv6* プロトコルの使用

説明	フォーマット	値
ソケット記述子	フルワード 2 進	**TAKESOCKET** コマンドで指定されるソケット番号
クライアント ID	40 バイト	リスナーのクライアント ID
クライアント・データ	35 バイト	リスナーに読み取られる最初の 35 バイト (存在する場合)
Filler	1 バイト	充てん文字
クライアントの SOCKADDR	2 バイトのファミリー 2 バイトのポート 4 バイトのフロー情報 16 バイトの IPv6 アドレス 4 バイトのスコープ	フロー情報とスコープは z/VSE では使用しません
	68 バイト	予約済み
データ長	ハーフワード 2 進	クライアントから受け取ったデータの長さ
データ域	前のフィールドで定義された長さ	位置 1 から始まる、クライアントから受け取ったデータ

リスナー用のユーザー独自のセキュリティー・リンク・モジュールの作成

リスナー・プロセスには、CICS トランザクションが開始される前にセキュリティー検査を実行するモジュールを作成し、それを組み込みたいユーザー向けに、出口点

が備わっています。出口点がインプリメントされているのは、モジュールが用意されない場合に、すべての有効なトランザクションが開始されるようにするためです。

セキュリティー・モジュールを作成する場合、そのモジュールを構成データ・セット内に定義しさえすれば、どのような名前を付けてもかまいません。このプログラムは、COBOL、PL/I、またはアセンブラー言語で作成することができ、それに適した項目を CICS プログラム処理テーブル (PPT) に入れる必要があります。

EZAC 内での指定**:** セキュリティー・モジュールの名前を、Alter または Define の SECexit フィールドに指定します。モジュールの名前を指定しない場合、CICS はモジュールがないものと想定します。詳しくは、標準リスナーの画面 2 を調べてください。

タスク作成プロセスの直前に、リスナーは、COMMAREA を渡す条件付き CICS LINK を使用して、セキュリティー・モジュールを起動します。リスナーは、セキュリティー検査用にモジュールが使用する情報と 1 バイトのスイッチとが含まれているデータ域を、モジュールに渡します。ユーザーのセキュリティー・モジュールは、セキュリティー検査と、それに合わせたスイッチの設定を実行する必要があります。

セキュリティー・モジュールが戻ると、リスナーは、スイッチの状態をチェックし、セキュリティーに問題がないことをスイッチが表している場合、トランザクションを開始します。モジュールは、CICS 環境で有効な任意の機能を実行できます。ただし、過剰な処理はパフォーマンス低下の原因になる場合があります。

表 27 に、セキュリティー・モジュールによって使用されるデータ域を示します。

表 *27. EZA* リスナーからセキュリティー/トランザクション出口への出力

説明	オフセット	フォーマット	値
CICS トランザクション ID	+000 (x00)	4 バイト文字	クライアントによって要求された CICS トランザクション
データ域	+004 (x04)	35 バイト文字	クライアントが受け取ったユーザー・データ
フォーマットの切り替え	+039 (x27)	1 バイト文字	フォーマットの切り替え (1= 拡張、0= 拡張しない)
	+040 (x28)	4 バイト	予約済み
アクション	+044 (x2C)	2 バイト文字	始動タスクの方式 (IC= インターバル制御、KC= タスク制御、TD= 一時データ)
インターバル制御時間	+046 (x2E)	6 バイト文字	IC 開始に要求されるインターバル (hhmmss)
アドレス・ファミリー	+052 (x34	ハーフワード 2 進	アドレス・ファミリー (2= AF_INET、19= AF_INET6)
クライアントのポート	+054 (x36)	ハーフワード 2 進	クライアントのポートのポート番号
クライアントの IPv4 アドレス	+056 (x38)	フルワード 2 進	クライアントの IPv4 アドレス

表 27. *EZA* リスナーからセキュリティー/トランザクション出口への出力 (続き)

説明	オフセット	フォーマット	値
スイッチ	+060 (x3C)	1 バイト文字	出口により設定されるスイッチ (1= トランザクションを許可、1 以外 = トランザクションを禁止)
スイッチ 2	+061 (x3D)	1 バイト文字	出口により設定されるスイッチ (1= リスナーがクライアントにメッセージを送信 1 以外 = 出口がクライアントにメッセージを送信)
端末識別名	+062 (x3E)	4 バイト文字	出口により設定：新規タスクに関連する端末がない場合、2 進ゼロを返します。そのほかの場合は、新規タスクに関連する CICS 端末 ID を返します。
ソケット記述子	*066 (x42)	ハーフワード 2 進	現行のソケット記述子
ユーザー ID	+068 (x44)	8 バイト文字	サーバー・トランザクションの開始で使用されるユーザー ID 値。これは、端末 ID とは相互に排他的です。
リスナーの IPv4 アドレス	+076 (x4C)	フルワード 2 進	ローカル IPv4 アドレス
リスナーのポート	+080 (x50)	ハーフワード 2 進	リスナーのポート番号
リスナーの IPv4 アドレス	+082 (x52)	16 バイトのバイナリー	ローカル IPv6 アドレス
リスナーのスコープ ID	+098 (x62)	フルワード 2 進	リクエスターの IPv6 アドレスのスコープ ID (VSE では使用されません)
クライアントの IPv6 アドレス	+102 (x66)	16 バイトのバイナリー	リクエスターのホストの IPv6 アドレス
クライアントのスコープ ID	+118 (x76)	フルワード 2 進	リクエスターの IPv6 アドレスのスコープ ID (VSE では使用されません)
クライアントの証明書の長さ	+118 (x7A)	ハーフワード 2 進	VSE ではサポートされない
クライアントの証明書アドレス	+124 (x7C)	フルワード 2 進	VSE ではサポートされない
	+128 (x80)	34 バイト	予約済み
データ長	+162 (xA2)	ハーフワード 2 進	クライアントから受け取ったデータの長さ
データ域	+164 (xA4)	前のフィールドで定義された長さ	位置 1 から始まる、クライアントから受け取ったデータ (拡張リスナーの場合は 35 バイトまでは最初のデータ域と同じです)

データ変換ルーチン

CICS は EBCDIC データ・フォーマットを使用しますが、TCP/IP ネットワークは ASCII を使用します。CICS と TCP/IP ネットワークとの間でデータを移動する際、アプリケーション・プログラムは必要なデータ変換を開始する必要があります。CICS プログラムは、z/VSE が提供する、以下を行うルーチンを使用できます。

- SEND、RECEIVE、READ、および WRITE 呼び出しで TCP/IP ネットワークとデータをやり取りするときに、EBCDIC と ASCII 間の変換を行う。
- SELECT 呼び出しを使用するときに、ビット配列と文字ストリング間の変換を行う。

詳細は、 323 ページの『ソケット呼び出しインターフェース用のデータ変換プログラムの使用』を参照してください。

第 6 部 付録

付録 A. TCP/IP for z/VSE での使用例

自律型 FTP

通常の環境では、VSE FTP デーモンがすべてのファイル転送を実行します。この理由のため、すべてのファイルは TCP/IP for z/VSE パーティションに定義されている必要があります。

環境によっては、これが不便な場合があります。例えば、バッチ処理が新規ファイルを作成し、そのファイルがリモート・ワークステーションに送信される場合、その新規ファイルを定義するために TCP/IP for z/VSE との対話が必要です。そのようにオペレーター介入を強制する代わりに、拡張 FTP コマンドが用意されています。それを使用すれば、TCP/IP パーティションがデータ・セットに関する情報をあらかじめ保有せずに、ローカルに定義された DLBL を指定することが可能です。この動作モードは、「自律型 FTP」であると考えることができます。

このモードでファイルを転送するには、次の形式でコマンドを使用します。

```
PUT %dlbl,type,recfm,lrecl,blksize  filespec
GET filespec  %dlbl,type,recfm,lrecl,blksize
```

パーセント記号 (%) は、ファイル名ではなく DLBL が指定済みであることを示します。他のパラメーターは、次のとおりです。

filespec
: リモート・システム上のファイル名。

type
: ファイル・タイプ。

recfm
: ファイルのレコード・フォーマット。

lrecl
: ファイルの論理レコード長。

blksize
: ファイルのブロック・サイズ。

自律型 FTP に使用されるすべてのパラメーターの詳しい説明については、「*TCP/IP for VSE User Guide*」を参照してください。

例

以下の例では、SAM-ESDS 作業ファイル 'A.KRUS.X1' ('X1' はパーティション ID から導かれる) が間接的に IDCAMS REPRO を介して定義されます。このファイルは、自律型 FTP を介してワークステーションに転送され、処理が正常に終了した後、IDCAMS によって削除されます。この方法の利点は、実際のファイルを明示的に定義する必要がなく、ファイル名を記憶しておく必要もないことです。

```
* $$ JOB JNM=FTPAUTNP,CLASS=X,DISP=D
// JOB  FTPAUTNP   TEST   AUTONOMIOUS FTP BATCH
// DLBL TESTNKD,'%A.KRUS',0,VSAM,CAT=ESCAT1,RECSIZE=120,              X
               DISP=(NEW,KEEP,DELETE),RECORDS=(150,100)
// DLBL TEST,'%A.KRUS',0,VSAM,CAT=ESCAT1
// DLBL COPYIN,'KRUS.SAMF',,VSAM,CAT=ESCAT1
// LOG
```

```
*
// EXEC IDCAMS,SIZE=AUTO
 REPRO INFILE (COPYIN) -
       OUTFILE (TESTNKD ENV(BLKSZ(120) RECFM(F))) -
       NOREUSE
/*
*
// EXEC FTP,PARM='IP=KRUSE'
KRUS
DAGI
DD
DD
LCD ESCAT1
CD VSE230/TEMP
PUT %TEST,SAM,F,120 FTPPUT.X1
QUIT
/*
IF $RC > 0 THEN
GOTO $EOJ
// EXEC IDCAMS,SIZE=AUTO
 DELETE (%A.KRUS                                ) -
     CLUSTER        -
     PURGE -
     CATALOG (ESCAT1.USER.CATALOG                        )
/*
/&
* $$ EOJ
```

注:

1. ジョブ・リスト中の作業ファイルは動的な名前を持っています。ここでは、次のとおりです。

   ```
   PUT %X1SAM,SAM,F,120 FTPPUT.X1
   ```

2. 自律型 FTP では以下は使用できませんが、FTPBATCH プログラムでは使用できます。
   - DISP オプションを指定した DLBL ステートメント
   - SAM-ESDS '%%working' ファイル

AUTOLPR - CICS レポート・コントローラー・フィーチャー (RCF) を使用した印刷

このセクションでは、CICS レポート・コントローラー・フィーチャー (RCF) を TCP/IP for z/VSE AUTOLPR フィーチャーと一緒に使用する例を示します。

TCP/IP for z/VSE は、LPR クライアント・アプリケーション、または AUTOLPR フィーチャーを使用するバッチからの印刷に加え、CICS RCF で生成されたファイルの自動印刷もサポートします。バッチからの印刷と同様に、VSE/POWER ユーザー情報フィールドまたは DEFINE EVENT 定義の HOSTNAME パラメーターに、スクリプト・ファイルのファイル名を指定する必要があります。このスクリプト・ファイルは、LPD (ライン・プリンター・デーモン) をホスティングするシステムのリモート IP アドレスと、指定されたホスト上の印刷を行うプリンターの名前を指定しなければなりません。

これらの情報が適切に指定されていると、TCP/IP for z/VSE は、指定された宛先に VSE/POWER リスト待ち行列の印刷出力を送信します。その際、EVENT (このセ

クションの下にある例を参照してください) が、指定された VSE/POWER クラスをカバーする TCP/IP for z/VSE に定義済みであると想定されます。

AUTOLPR に関する詳しい説明は、「*TCP/IP for VSE User Guide*」を参照してください。

CICS RCF プログラム内での指定

CICS RCF プログラム内には、VSE/POWER クラスと、ユーザー情報フィールド内のスクリプト・ファイルの名前を指定する必要があります。これらの必須値は、VSE/POWER に渡されます。

以下の例の場合、これらの値は次のとおりです。

- VSE/POWER クラスは `CLASS('T')`
- ユーザー情報は `USERDATA(SCRIPTNM)`

これらの値は、ユーザー要件に合わせて違う値にしてもかまいません。

```
   ...
DFHEISTG DSECT
SCRIPTNM DS    CL16
   ...
TESTLPR  CSECT
* Open output spoolfile
         MVC   SCRIPTNM,=CL16'SCRIPT2'   Set Script Name
*    Script-Name-Field should be 16 characters long
*    Script-Name-Field should be padded with blanks
         EXEC CICS SPOOLOPEN REPORT('LPRTEST') USERDATA(SCRIPTNM)   *
               TOKEN(OUTTOKEN) NOCC CLASS('T') NOSEP                *
               RESP(RESPFLD) RESP2(RESP2FLD)
   ...
```

TCP/IP 定義

TCP/IP for z/VSE 構成ファイル IPINITxx.L には、次の (またはこれに似た) 定義が含まれている必要があります。スタートアップ構成にこれらを定義しなかった場合は、対話式に TCP/IP for z/VSE に定義することもできます。

- AUTOLPR for VSE/POWER LST 待ち行列の定義 `CLASS T`

  ```
  DEFINE EVENT,ID=LPR,TYPE=POWER,CLASS=T,QUEUE=LST
  ```

- IP アドレス `9.1.2.3` のシンボル名 `REMHOST`

  ```
  DEFINE NAME,NAME=REMHOST,IPADDR=9.1.2.3
  ```

- VSE ライブラリー・メンバー `PRTLOCAL.L` に支えられた、スクリプト `SCRIPT2` のスクリプト・ファイル定義

  ```
  DEFINE NAME,NAME=SCRIPT2,SCRIPT=PRTLOCAL
  ```

スクリプト・ファイル定義

スクリプト・ファイルは、VSE ライブラリー内の L ソース・ブックとしてカタログされている必要があり、**// LIBDEF SOURCE,SEARCH** チェーンを介してアクセス可能である必要があります。前述の例では、メンバー名は `PRTLOCAL.L` です。スクリプト・ファイルには、必要なホスト定義およびプリンター定義が含まれています。

```
* $$ JOB JNM=CATAL,CLASS=A,DISP=D
// JOB CATAL  CATALOG SCRIPT MEMBER PRTLOCAL.L
// EXEC LIBR
ACC S=PRD2.CONFIG
CAT PRTLOCAL.L  R=Y
SET HOST=REMHOST         SYMBOLIC HOST NAME
SET PRINTER=PRINTER1
/+
/*
/&
* $$ EOJ
```

GPS および RCF

以下の例は、CICS のレポート・コントローラー・フィーチャー (RCF) による GPS の使用のために、TCP/IP-GPS、VTAM、および CICS に対して行うべき定義を示します。

GPS デーモンを定義するためのすべてのパラメーターの詳細については、「*TCP/IP for VSE Optional Features*」を参照してください。

VTAM に対する定義

```
TCPPRT   VBUILD TYPE=APPL
GPS1     APPL  AUTH=(ACQ),DLOGMOD=DSC2K
GPS2     APPL  AUTH=(ACQ),DLOGMOD=DSC2K
```

注: 1 つの VTAM プリンターを複数の異なる CICS アプリケーションで共用しなければならない場合、プリンターは、まず 1 つの CICS から解放されてからでないと、別の CICS で使用することはできません。これは、そのプリンターの CICS TYPETERM 定義に RELREQ=YES を定義することで行われます。ただし、RELREQ=YES をアクティブにするため、VTAM APPL ステートメントに SESSLIM=YES がコーディングされていなければなりません。詳細については、「*VTAM Programming Guide*」を参照してください。

CICS に対する定義

```
CEDA DEFine TYpeterm:  GPSPRT   Group:  VSETERM1

CEDA DEFine TErminal:  GPS1     Group:  VSETERM1
CEDA DEFine TErminal:  GPS2     Group:  VSETERM1
```

TCP/IP に対する定義

```
DEFINE FILE,PUBLIC='PRD2.GPSWORK',DLBL=PRD2,TYPE=LIBRARY
*
* GPS1 is a IBM4248
DEFINE GPSD,ID=GPS001,STORAGE='PRD2.GPSWORK',TERMNAME=GPS1,-
IPADDR=nnn.nnn.nnn.nnn,PRINTER=LOCAL
*
* GPS2 is a IBM3130
DEFINE GPSD,ID=GPS002,STORAGE='PRD2.GPSWORK',TERMNAME=GPS2,-
IPADDR=nnn.nnn.nnn.nnn,PRINTER=printername
```

'printername' は大/小文字の区別があることに注意してください。

RCF に対する定義

```
PRINTER             DESTINATION
  GPS1                  GPS1
  GPS2                  GPS2
```

TELNET と Class-C ネットワークでのサブネット分割

以下の例は、Class-C ネットワークを分割して Telnet で使用するためのいくつかのサブネットを作成する方法を示します。これは、それぞれのサブネットに対して異なるサブネット・マスクを使用することで行われます。

要件/質問

```
CICS Terminal Id = TA31xx   ->  IPaddress 9.222.66.1   -  27
                 = TA03xx   ->  IPaddress 9.222.66.65  -  91
                 = TA06xx   ->  IPaddress 9.222.66.129 - 155
```

各端末 ID がそれぞれ異なるようにして各ユーザーを識別可能にするには、どのようにすればいいでしょうか?

回答

```
DEFINE MASK,ID=net1mask,NETWORK=9.222.66.0,MASK=255.255.255.224
DEFINE MASK,ID=net2mask,NETWORK=9.222.66.64,MASK=255.255.255.224
DEFINE MASK,ID=net3mask,NETWORK=9.222.66.128,MASK=255.255.255.224
DEFINE TELNETD,ID=teln1,MENU=MENU3,COUNT=30,TERMNAME=TA31, -
                        IPADDR=9.222.66.0
DEFINE TELNETD,ID=teln2,MENU=MENU4,COUNT=30,TERMNAME=TA03, -
                        IPADDR=9.222.66.64
DEFINE TELNETD,ID=teln3,MENU=MENU5,COUNT=30,TERMNAME=TA06, -
                         IPADDR=9.222.66.128
```

TELNET デーモンとログモード

次の例は、TELNET セッションを照会可能にする方法を示します。

TELNET デーモンの定義が次のようになっているとします。

```
DEFINE TEL,ID=MYTEL,TAR=DBDCCICS,TERM=T1000,CO=20,LOGMODE=SP3272QN, -
       LOGMODE3=SP3272QN,LOGMODE4=SP3272QN,LOGMODE5=SP3272QN
```

この場合、すべてのタイプの端末 (モデル 2、3、4、および 5) が照会可能になります。設定されているのが、

```
LOGMODE=SP3272QN
```

だけの場合、モデル 3 は SP3272QN ではなく、拡張データ・ストリームなしのデフォルト値 D4B32783 になります。このため、上の定義をお勧めします。照会可能なセッションが必要でない場合は、EXTDS を持つ IUI デフォルトのログモードは次のようになります。

```
DEFINE TEL,ID=MYTEL,TAR=DBDCCICS,TERM=T1000,CO=20,LOGMODE=SP3272EN, -
       LOGMODE3=SP3273EN,LOGMODE4=NSX32704,LOGMODE5=NSX32705
```

IUI においてモデル 4 と 5 には明示的なログモードがないため、VTAM デフォルトが使用されます。

VSAMCAT の使用法

FTP または HTTP を介してアクセスしようとしているすべての VSAM ファイルを定義するのではなく、VSAM カタログを TCP/IP for z/VSE に定義し、カタログ内のすべてのクラスターについて DLBL とファイル制御ブロック情報が動的に構築されるようにすることができます。

DEFINE FILE コマンドの VSAMCAT パラメーターの詳しい説明は、「*TCP/IP for VSE Command Reference*」を参照してください。

1. VSE に対してカタログを定義する

   VSAM カタログを使用する最初のステップは、カタログを定義する DLBL を用意することです。VSAMCAT fileIO ドライバーは、クラスター属性情報を入手するため、カタログを順次に読み取ります。そのため、DLBL には、それ自体を指す「,CAT=」パラメーターがなければなりません。カタログ IJSYSUC を例にとると次のようになります。

   ```
   // DLBL IJSYSUC,'VSAM.USER.CATALOG',,VSAM,CAT=IJSYSUC
   ```

   標準ラベル中の項目を修正するか、新規項目を作成してそれを TCP/IP スタートアップ JCL に入れるかのいずれかの方法が可能です。どちらの場合も、TCP/IP がカタログの DLBL を検出し、カタログ項目にそれ自体を指す ",CAT=" が指定されていることが重要です。

2. TCP/IP に対してカタログを定義する

   この時点で VSE にはカタログのことが分かっているので、次は TCP/IP に知らせます。同じカタログについてのサンプル定義を以下に示します。

   ```
   DEFINE FILE,PUBLIC='IJSYSUC',DLBL=IJSYSUC,TYPE=VSAMCAT
   ```

   パブリック名は任意の名前にできますが、この例では、DLBL 名と同じ名前にします。

3. カタログを使用する

   この時点で VSE および TCP/IP システムにカタログについて知らせたので、それが実際に存在している場合、IJSYSUC (この例の場合) への「ChDir」の発行によって、そのカタログに FTP でアクセスできます。初めてこれを実行するときには、パーティション・ストレージに fileIO モジュール IPNFVCAT がロードされたことを示すメッセージが SYSLOG に出力されます。DirList を実行すると、IPNFVCAT はカタログ情報を読み取り、情報のリストを戻します。特定の項目に対して RETRieve を発行すると、IPNFVCAT はパーティションをチェックして、既に存在する DLBL があるかどうかを調べます。ない場合は、動的にパーティションに追加されます。その後、ファイルが転送されます。

   これの唯一の例外は、VSAM が制御する SAM ファイルです。VSAM カタログはレコード数などの情報で更新されないので、これらのファイルを VSAMCAT を使用して取得することはできません。このような場合、これらのファイルを個別に TCP/IP に「TYPE=SAM」として定義し、DTFSD 方法を使用して取得する必要があります。

   VSAM カタログへの PUT の実行は異なります。FTP の場合、クラスターが既に定義済みであるようにするか、クラスターが動的に定義されるようにするか、ファイルを定義するための REXX プログラムが FTP から実行されるようにする必要があります。

   最後に、VSAM ファイルに対して DELEte を実行できます。そうすると、IDCAMS が動的に起動されて DELETE CLUSTER が実行されます。ただし、VSAMCAT ファイルには RENAME は機能しません。

コマンド・プリプロセッサーの使用

EXEC TCP ベースのプログラムは、言語特有のコード構造を生成するため、TCP/IP for z/VSE プリプロセッサー・プログラム IPNETPRE を必要とします。

IPNETPRE を実行するときには、EXEC ステートメントの PARM フィールドを使用して 2 つのオプションを指定します。次に例を示します。

```
// EXEC IPNETPRE,SIZE=IPNETPRE,PARM='LANG=COBOL,ENV=CICS'
* $$ SLI MEM=COBSRC.C,S=PRD3.INGO
/*
```

LANG

LANG=xxx パラメーターは、処理される言語をプリプロセッサーに知らせます。xxx にサポートされている値は、次のとおりです。

ASSEMBLER
High-Level Assembler

COBOL
COBOL for VSE

PL1 PL/I for VSE

ENV **ENV=xxx** パラメーターは、完成したプログラムが実行する環境を示します。xxx に許容される値は、次の 2 つです。

BATCH
プログラムはバッチ・モードで実行します。

CICS プログラムは CICS 下で実行されます。

注:

1. ENV=CICS の場合、プログラムは常に TCP/IP プリプロセッサーを実行した後で CICS プリプロセッサーを実行します。TCP/IP プリプロセッサーは EXEC CICS ステートメントを生成し、それらは CICS プリプロセッサーで置換されるので、この実行順を守らなければなりません。

TCP/IP for z/VSE プリプロセッサーについて詳しくは、「*TCP/IP for VSE Programmer's Guide*」を参照してください。

サンプル・プログラム

以下に示すサンプル・プログラムは、同じ機能をさまざまな言語で表したものです。どの場合も、データ操作の「特殊な」手法に注意してください。

COBOL の例

```
IDENTIFICATION DIVISION.

  PROGRAM-ID.        COBSRC.
  AUTHOR.            JOHN DOE.
      INSTALLATION.  WORTHINGTON OHIO.
      DATE-WRITTEN.  AUGUST 2, 1995.
  DATE-COMPILED.

  ENVIRONMENT DIVISION.

  CONFIGURATION SECTION.
```

```
 SOURCE-COMPUTER.   IBM-370.
 OBJECT-COMPUTER.   IBM-370.

 DATA DIVISION.

 EXEC TCP CONTROL DOUBLE(NO)
 END-EXEC.

 WORKING-STORAGE SECTION.
 01  WORK-AREA-ONE.
     05  PART1       PICTURE 9(4) COMP.
     05  PART2       PICTURE 9(4) COMP.
     05  PART3       PICTURE 9(4) COMP.
     05  PART4       PICTURE 9(4) COMP.
     05  IPADDRESS.
         10 IPAD1    PICTURE X.
         10 IPAD2    PICTURE X.
         10 IPAD3    PICTURE X.
         10 IPAD4    PICTURE X.
     05  HALFWORD    PICTURE 9(4) COMP.
     05  HALFWORD-X  REDEFINES HALFWORD.
         10 BYTEX1   PICTURE X.
         10 BYTEX2   PICTURE X.
     05  RESULTS.
         10 RECB     PICTURE X(4).
         10 RLOPORT  PICTURE 9(4) COMP.
         10 RFOPORT  PICTURE 9(4) COMP.
         10 RFOIP    PICTURE X(4).
         10 RCOUNT   PICTURE 9(4) COMP.
         10 RFLAGS   PICTURE X.
         10 RCODE    PICTURE X.
         10 RTERMTY  PICTURE X(40).
     05  MY-DESC     PICTURE X(4).
 01  LOCAL-PORT      PICTURE 9(4) COMP.
 01  BUFFER.
     05  WORKAREA    PICTURE X(512).
 PROCEDURE DIVISION.

 BEGIN.
*---------------------------------------*
*                                       *
*            First Test                 *
*                                       *
*---------------------------------------*
*
*    Setup IPADDRESS to hold 172.20.10.10 in binary
*
     MOVE 172 TO HALFWORD.
     MOVE BYTEX2 TO IPAD1.
     MOVE 20  TO HALFWORD.
     MOVE BYTEX2 TO IPAD2.
     MOVE 10  TO HALFWORD.
     MOVE BYTEX2 TO IPAD3.
     MOVE 10  TO HALFWORD.
     MOVE BYTEX2 TO IPAD4.
*
*    Attempt to open a connection at 172.20.10.10 port 2000
*
 EXEC TCP OPEN FOREIGNPORT(2000)
               FOREIGNIP(IPADDRESS)
               LOCALPORT(0)
               RESULTAREA(RESULTS)
               DESCRIPTOR(MY-DESC)
               ACTIVE
               WAIT(YES)
               ERROR(SECOND-TEST)
 END-EXEC.
```

```
     DISPLAY 'Open has completed'.
*
*    Receive a piece of data
*
 EXEC TCP RECEIVE
               TO(BUFFER)
               LENGTH(512)
               RESULTAREA(RESULTS)
               DESCRIPTOR(MY-DESC)
               WAIT(YES)
               ERROR(SECOND-TEST)
 END-EXEC.
     DISPLAY 'Receive has completed'.
*
*    Close the connection
*
 EXEC TCP CLOSE
               RESULTAREA(RESULTS)
               DESCRIPTOR(MY-DESC)
               WAIT(YES)
               ERROR(SECOND-TEST)
 END-EXEC.
     DISPLAY 'Close has completed'.
*--------------------------------------*
*                                      *
*            Second Test               *
*                                      *
*--------------------------------------*
 SECOND-TEST.
*
*    Attempt to open another connection
*
     MOVE 2000 TO LOCAL-PORT.
 EXEC TCP OPEN FOREIGNPORT(0)
               FOREIGNIP(0)
               LOCALPORT(LOCAL-PORT)
               RESULTAREA(RESULTS)
               DESCRIPTOR(MY-DESC)
               PASSIVE
               WAIT(YES)
               ERROR(ERROR-SPOT)
 END-EXEC.
     DISPLAY 'Second Open has completed'.
*
*    Display the foreign IP address
*
     MOVE RFOIP TO IPADDRESS.
     MOVE IPAD1 TO BYTE2.
     MOVE HALFWORD TO PART1.
     MOVE IPAD2 TO BYTE2.
     MOVE HALFWORD TO PART2.
     MOVE IPAD3 TO BYTE2.
     MOVE HALFWORD TO PART3.
     MOVE IPAD4 TO BYTE2.
     MOVE HALFWORD TO PART4.
     DISPLAY PART1 '.' PART2 '.' PART3 '.' PART4
*
*    Send another piece of data
*
 EXEC TCP SEND
               FROM(BUFFER)
               LENGTH(512)
               RESULTAREA(RESULTS)
               DESCRIPTOR(MY-DESC)
               WAIT(YES)
               ERROR(ERROR-SPOT)
 END-EXEC.
```

```
           DISPLAY 'Second Send has completed'.
      *
      *    Close the second connection
      *
       EXEC TCP CLOSE
                     RESULTAREA(RESULTS)
                     DESCRIPTOR(MY-DESC)
                     WAIT(YES)
                     ERROR(ERROR-SPOT)
       END-EXEC.
           DISPLAY 'Second Close has completed'.

           STOP RUN.

       ERROR-SPOT.

           STOP RUN.
```

PL/I の例

```
SAMPLE4: PROCEDURE OPTIONS(MAIN);

 DCL IPADDRESS     BINARY FIXED(31,0);
 DCL MY-DESC       CHAR(4);
 DCL 1  RESULTS,
        2 RECB     CHAR(4),
        2 RLOPORT  BINARY FIXED(15,0),
        2 RFOPORT  BINARY FIXED(15,0),
        2 RFOIP    CHAR(4),
        2 RCOUNT   BINARY FIXED(15,0),
        2 RFLAGS   CHAR(1),
        2 RCODE    CHAR(1),
        2 RTERMTY  CHAR(40);
 DCL MY-DESC       CHAR(4);
 DCL LOCAL-PORT    BINARY FIXED(15,0);
 DCL BUFFER        CHAR(512);

/*--------------------------------------*
 *                                      *
 *            First Test                *
 *                                      *
 *--------------------------------------*/
/*
 *    Attempt to open a connection at 172.20.10.10 port 2000
 */
       EXEC TCP OPEN FOREIGNPORT(2000)
                     FOREIGNIP(IPADDRESS)
                     LOCALPORT(0)
                     RESULTAREA(RESULTS)
                     DESCRIPTOR(MY-DESC)
                     ACTIVE
                     WAIT(YES)
                     ERROR(SECOND-TEST);
/*
 *    Receive a piece of data
 */
       EXEC TCP RECEIVE
                     TO(BUFFER)
                     LENGTH(512)
                     RESULTAREA(RESULTS)
                     DESCRIPTOR(MY-DESC)
                     WAIT(YES)
                     ERROR(SECOND-TEST);
/*
 *    Close the connection
 */
       EXEC TCP CLOSE
```

```
                    RESULTAREA(RESULTS)
                    DESCRIPTOR(MY-DESC)
                    WAIT(YES)
                    ERROR(SECOND-TEST);
SECOND-TEST:
/*--------------------------------------*
 *                                      *
 *            Second Test               *
 *                                      *
 *--------------------------------------*
 *
 *    Attempt to open a connection
 */
          LOCAL-PORT = 2000;
       EXEC TCP OPEN FOREIGNPORT(0)
                    FOREIGNIP(0)
                    LOCALPORT(LOCAL-PORT)
                    RESULTAREA(RESULTS)
                    DESCRIPTOR(MY-DESC)
                    PASSIVE
                    WAIT(YES)
                    ERROR(ERROR-SPOT);
/*
 *    Display the foreign IP address
 */

/* Need code here..... */

/*
 *    Receive a piece of data
 */
       EXEC TCP SEND
                    FROM(BUFFER)
                    LENGTH(512)
                    RESULTAREA(RESULTS)
                    DESCRIPTOR(MY-DESC)
                    WAIT(YES)
                    ERROR(ERROR-SPOT);
/*
 *    Close the connection
 */
       EXEC TCP CLOSE
                    RESULTAREA(RESULTS)
                    DESCRIPTOR(MY-DESC)
                    WAIT(YES)
                    ERROR(ERROR-SPOT);

 RETURN;
 END SAMPLE4;
```

プログラムのコンパイル

アプリケーション・プログラムのコーディングが終わり、プリプロセッサーの処理が完了したら、適切なコンパイラーにサブミットする必要があります。多くの場合、プリコンパイラーからの出力を 1 つ以上の別のプリコンパイラーに渡す必要もあります。以下の例で、これを行う 1 つの方法を示します。例では、 607 ページの『COBOL の例』 の COBOL の例 COBSRC.C を使用します。

バッチ用に COBOL プログラムをコンパイルする

最初の例は、ライブラリー PRD3.INGO に保管されたソース COBSRC.C (607 ページの『COBOL の例』 を参照) のコンパイル方法を示します。これによってフェーズ SAMPLEB が生成されます。

ステップ 1 - メイン・ジョブ

メイン・ジョブの構造は、バッチ環境の場合でも CICS ランタイム環境の場合でも同じです。PRD3.INGO に保管された COMSTP1.PROC プロシージャーが呼び出されます。

```
* $$ JOB JNM=COMPILE,CLASS=4,DISP=D
* $$ LST CLASS=W,DISP=D
* $$ PUN CLASS=4,DISP=I
// JOB COMPILE TCPIP PROGRAM
// LIBDEF *,SEARCH=(PRD3.INGO,PRD2.TCPIPC,PRD1.BASE,PRD2.PROD,PRD2.SCEEBASE)
// EXEC PROC=COMSTP1
/&
* $$ EOJ
```

ステップ 2 - EXEC TCP ステートメントの処理

プロシージャー COMSTP1.PROC は TCP/IP プリプロセッサー **IPNETPRE** を呼び出し、ユーティリティー・プログラム **IESINSRT** を使用して新しいジョブを生成します。この新規ジョブは CATAL1 という名前で、TCP/IP プリプロセッサー出力を PRETCP.DAT という名前でカタログして PRD3.INGO に入れるためのものです。この後、さらに処理を行うため COMSTP2.PROC が呼び出されます。

```
// ASSGN SYSIPT,SYSRDR
// EXEC IESINSRT
$ $$ JOB JNM=CATAL1,CLASS=4,DISP=D
$ $$ LST CLASS=W,DISP=D
$ $$ PUN CLASS=4,DISP=I
// JOB CATAL1 CATALOG OUTPUT OF THE TCPIP preprocessor
// LIBDEF *,SEARCH=(PRD3.INGO,PRD2.TCPIPC,PRD1.BASE,PRD2.PROD,PRD2.SCEEBASE)
// LIBDEF *,CATALOG=PRD3.INGO
// EXEC LIBR
   ACC S=PRD3.INGO
   CATALOG PRETCP.DAT  EOD=/(    REPLACE=YES
* $$ END
// OPTION DECK
*
* Process EXEC TCP source for CICS
*
// EXEC IPNETPRE,SIZE=IPNETPRE,PARM='LANG=COBOL,ENV=BATCH'
* $$ SLI MEM=COBSRC.C,S=PRD3.INGO
/*
// EXEC IESINSRT
/(
/*
// EXEC PROC=COMSTP2
#&
$ $$ EOJ
* $$ END
```

ステップ 3 - コンパイルおよびリンク・エディット

プロシージャー COMSTP2.PROC は **COBOL for VSE** コンパイラーを起動し、リンケージ・エディターを呼び出して、コンパイラーが生成した OBJ デックをリンクします。結果のフェーズ SAMPLEB は PRD3.INGO に保管されます。なぜなら、

```
// LIBDEF *,CATALOG=PRD3.INGO
```

がまだアクティブだからです。

```
*
* Compile and link phase SAMPLEB for Batch
*
```

```
// OPTION CATAL
   PHASE SAMPLEB,*
// EXEC IGYCRCTL,SIZE=IGYCRCTL
   CBL TEST APOST
* $$ SLI MEM=PRETCP.DAT,S=PRD3.INGO
/*
// EXEC LNKEDT
```

CICS 用に COBOL プログラムをコンパイルする

2 番目の例は、ライブラリー PRD3.INGO に保管されたソース COBSRC.C (607 ページの『COBOL の例』 を参照) のコンパイル方法を示します。これによってフェーズ SAMPLEC が生成されます。

ステップ 1 - メイン・ジョブ

メイン・ジョブの構造は、既にバッチ環境の場合に対して示したものと同じです。PRD3.INGO に保管された COMSTP1.PROC プロシージャーが呼び出されます。

```
* $$ JOB JNM=COMPILE,CLASS=4,DISP=D
* $$ LST CLASS=W,DISP=D
* $$ PUN CLASS=4,DISP=I
// JOB COMPILE TCPIP PROGRAM
// LIBDEF *,SEARCH=(PRD3.INGO,PRD2.TCPIPC,PRD1.BASE,PRD2.PROD,PRD2.SCEEBASE)
// EXEC PROC=COMSTP1
/&
* $$ EOJ
```

ステップ 2 - EXEC TCP ステートメントの処理

プロシージャー COMSTP1.PROC は TCP/IP プリプロセッサー **IPNETPRE** を呼び出し、ユーティリティー・プログラム **IESINSRT** を使用して新しいジョブを生成します。この新規ジョブは CATAL1 という名前で、TCP/IP プリプロセッサー出力を PRETCP.DAT という名前でカタログして PRD3.INGO に入れるためのものです。この後、さらに処理を行うため COMSTP2.PROC が呼び出されます。

```
// ASSGN SYSIPT,SYSRDR
// EXEC IESINSRT
$ $$ JOB JNM=CATAL1,CLASS=4,DISP=D
$ $$ LST CLASS=W,DISP=D
$ $$ PUN CLASS=4,DISP=I
// JOB CATAL1 CATALOG OUTPUT OF THE TCPIP preprocessor
// LIBDEF *,SEARCH=(PRD3.INGO,PRD2.TCPIPC,PRD1.BASE,PRD2.PROD,PRD2.SCEEBASE)
// LIBDEF *,CATALOG=PRD3.INGO
// EXEC LIBR
   ACC S=PRD3.INGO
   CATALOG PRETCP.DAT  EOD=/(    REPLACE=YES
* $$ END
// OPTION DECK
*
* Process EXEC TCP source for CICS
*
// EXEC IPNETPRE,SIZE=IPNETPRE,PARM='LANG=COBOL,ENV=CICS'
* $$ SLI MEM=COBSRC.C,S=PRD3.INGO
/*
// EXEC IESINSRT
/(
/*
// EXEC PROC=COMSTP2
#&
$ $$ EOJ
* $$ END
```

ステップ 3 - EXEC CICS ステートメントの処理

EXEC CICS ステートメントが TCP/IP プリプロセッサーによって生成されたので、CICS プログラミング・モデルに従って、適切な、例えばストレージの割り振りや WAIT などを行う CICS プリプロセッサーを、COBOL コンパイラーを呼び出す前に起動する必要があります。

既にステップ 2 で示したように、ここでも COMSTP2.PROC が動的に新しいジョブ CATAL2 を生成します。このジョブの目的は、CICS プリプロセッサーからの出力を PRECICS.DAT として保管し、次に、最後のコンパイルおよびリンク・エディットのステップのための COMSTP3.PROC を呼び出すことです。

```
// ASSGN SYSIPT,SYSRDR
// EXEC IESINSRT
$ $$ JOB JNM=CATAL2,CLASS=4,DISP=D
$ $$ LST CLASS=W,DISP=D
$ $$ PUN CLASS=4,DISP=I
// JOB CATALOG OUTPUT OF THE CICS preprocessor
// LIBDEF *,SEARCH=(PRD3.INGO,PRD2.TCPIPC,PRD1.BASE,PRD2.PROD,PRD2.SCEEBASE)
// LIBDEF *,CATALOG=PRD3.INGO
// EXEC LIBR
   ACC S=PRD3.INGO
   CATALOG PRECICS.DAT  EOD=/(    REPLACE=YES
* $$ END
*
* Starting CICS command preprocessor
*
// EXEC DFHECP1$,PARM='CICS'
* $$ SLI MEM=PRETCP.DAT,S=PRD3.INGO
/*
// EXEC IESINSRT
/(
/*
// EXEC PROC=COMSTP3
#&
$ $$ EOJ
* $$ END
```

ステップ 4 - コンパイルおよびリンク・エディット

プロシージャー COMSTP3.PROC は **COBOL for VSE** コンパイラーを起動し、リンケージ・エディターを呼び出して、コンパイラーが生成した OBJ デックをリンクします。結果のフェーズ SAMPLEC は PRD3.INGO に保管されます。なぜなら、

```
// LIBDEF *,CATALOG=PRD3.INGO
```

がまだアクティブだからです。

```
*
* Compile and link phase SAMPLEC for CICS
*
// OPTION CATAL
   PHASE SAMPLEC
   INCLUDE DFHELII
// EXEC IGYCRCTL,SIZE=IGYCRCTL
   CBL TEST APOST
* $$ SLI MEM=PRECICS.DAT,S=PRD3.INGO
/*
// EXEC LNKEDT
```

付録 B. EZASMI および EZASOKET インターフェースのデバッグ機能 (EZAAPI トレース)

EZA TCP/IP HLL API (EZASMI マクロ・インターフェースおよび EZASOKET 呼び出しインターフェース) には、トレース機能が組み込まれています。

このトレース機能は、TCP/IP for z/VSE および Linux Fast Path に加え、EZA Multiplexer および EZA OpenSSL のインターフェース・モジュールで使用可能です。これにより、EZASMI または EZASOKET ソケット呼び出しごとに 1 つ (または複数) のトレース・メッセージが生成されます。トレース機能では、これらの呼び出しのトレースを、システムのすべてのパーティションか、または、選択されたパーティションおよび動的クラスについて実行できます。トレース・メッセージは、SYSLOG または SYSLST に送信できます。

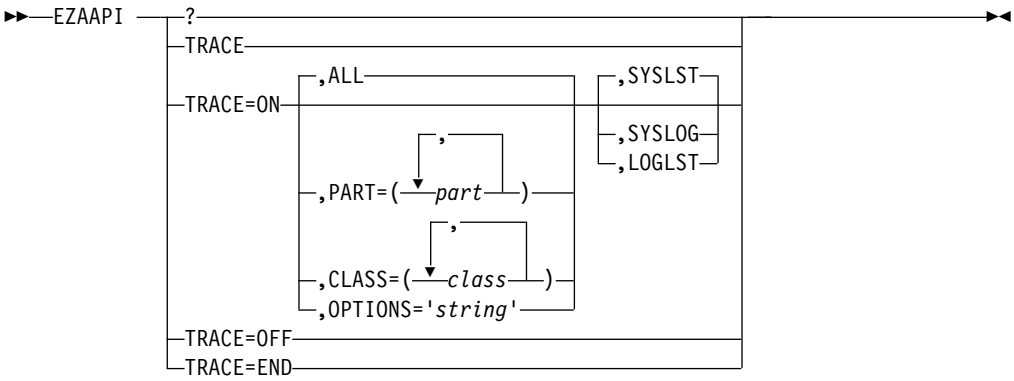
このトレース機能は単に「EZAAPI トレース」と呼ばれています。

使用要件

EZAAPI トレースは、モジュール EZASOHTR が SVA (z/VSE ではこれがデフォルト) にロードされていることを必要とします。

フォーマット

EZAAPI トレースは、AR コマンド EZAAPI を使用して、活動化および制御することができます。



パラメーター

EZAAPI ?
コマンド構文を表示します

EZAAPI TRACE
現在のトレース設定を表示します

EZAAPI TRACE=ON
定義して開始するか、再開します

(トレースが未定義の場合のデフォルト**)**
デフォルト ALL と SYSLST を使用し、トレースを定義して開始します

(EZAAPI TRACE=OFF 以降のデフォルト**)**
トレースを再開します

All システムのすべてのパーティションに対してトレースを定義して開始します

PART=(part,..)
選択されたパーティションに対してトレースを定義して開始します

CLASS=(class,..)
選択された動的クラス対してトレースを定義して開始します

OPTIONS='*string***'**
特殊なトレース・オプションを設定します。最大長 8 バイトの 16 進文字 *string* を使用できます。OPTIONS='' は、既存の OPTION の指定をリセットして消去します。OPTIONS ストリングが使用されるかどうかは、TCP/IP インターフェース・ルーチンによって異なります。適切な製品資料を参照してください。

EZAAPI TRACE=OFF
現在のトレースを中断します

EZAAPI TRACE=END
トレースを終了し、すべてのトレース定義をクリアします

SYSLST
トレース出力を SYSLST に送信します (SYSLST が割り当て済みの場合)

SYSLOG
トレース出力を SYSLOG に送信します

LOGLST
トレース出力を SYSLOG および SYSLST に送信します

出力

EZAAPI トレースが生成するメッセージは、読めば分かるような内容になっています。

EZAAPI トレースが開始すると、その次に行われる EZA インターフェースの呼び出し (EZASMI または EZASOKET) で以下のメッセージが発生します。

```
EZA001I EZASOH99 (Level *date*) started
```

EZA インターフェースの呼び出しごとに、次のメッセージが生成されます。

1. 次のような、1 つの開始メッセージ

   ```
   EZA002I >>> SOCKET processing starts ...
   ```

2. 追加の入力を示すいくつかのメッセージ

   ```
   EZA033I .. with AF/SOCTYPE/PROTO=02/01/00
   ```

3. 1 つ (または複数) の完了メッセージ

   ```
   EZA003I SOCKET returns with RC/ERRNO=00000/00000
   ```

EZA ソケット・アプリケーションとともに TCP/IP for z/VSE を使用する場合、EZAAPI トレースを活動化すると、TCP/IP for z/VSE の BSD-C トレース ($SOCKDBG トレースと呼ばれる) が自動的に活動化されます。

付録 C. 拡張 OSAX デバイス・ドライバーの構成

この付録では、z/VSE での拡張 OSAX デバイス・ドライバーの構成について概説します。

構成可能な QDIO バッファー数

QDIO 入力キュー・バッファー (入力バッファー) および出力キュー・バッファー (出力バッファー) の数は構成可能です。

z/VSE と Linux on z Systems 間の高速 TCP/IP 接続に HiperSockets ネットワークを使用して Linux on z Systems を活用することで z/VSE ソリューションを拡張するには、デフォルト値の変更が必要になる場合があります。再送信の必要なくデータを常に確実に送信できれば、最適なパフォーマンスを達成できます。HiperSockets はデータを同期で転送するので、データを確実に送信できるかどうかは、ターゲット・システムの空き QDIO (Queued Direct I/O) 入力バッファーの数によって決まります。z/VSE は、デフォルトで 8 つの入力バッファーを使用します。特に Linux on z Systems システムで入力バッファーの数を増加した場合には、この数は必ずしも十分ではありません。

OSAX デバイス・ドライバーは、8 つの出力バッファーのデフォルトで動作します。ワークロードの負荷が高い間は、すべての出力バッファーが使用されることがあり、データの再送信が必要になります。これはパフォーマンスに影響を与える可能性があり、メッセージ 0S39I および理由コード x'0026' が表示されます。 z/VSE は、最大 64 個の出力バッファーを構成できます。

HiperSockets (CHPID タイプ IQD) および OSA-Express デバイス (CHPID タイプ OSD および OSX) の入出力バッファーの数を構成する場合は、ICCF ライブラリー 59 の構成スケルトン SKOSACFG を使用できます。また、入力バッファーを追加する場合は、TCP/IP パーティションのサイズを大きくすることも必要になる可能性があります。

注: z/VSE では、デフォルトの入出力バッファーを使用する場合、1 つのリンクにつき 1 MB の 31 ビット・パーティションの GETVIS スペースが必要です。入力バッファーを追加するごとに、z/VSE はさらに 31 ビット・パーティションの GETVIS スペースを必要とします。ご使用の IOCDS 定義によって、OSA-Express には 64 K、および HiperSockets には最大 64 K です。そのため、入出力バッファーの数を増やす場合は、パーティションの GETVIS スペースの拡張も必要になる可能性があります。入力バッファーに対して PFIX が行われるため、TCP/IP 始動ジョブの **JCL SETPFIX LIMIT** ステートメントの値もそれに合わせて増やす必要があります。

VLAN サポート

仮想 LAN (VLAN) サポートにより、TCP/IP スタックで IPv4 または IPv6 の両方のレイヤー 2 またはレイヤー 3 リンクの特定の VLAN ID を登録できるようになります。VLAN を使用するようにシステムを構成するには、2 つの方法があります。

- TCP/IP を使用している場合は、IPv6/VSE **LINK** コマンドを使用します。詳しくは、次のサイトにある「IPv6/VSE Installation Guide」を参照してください。

  `http://www.ibm.com/systems/z/os/zvse/documentation/#tcpip`
- TCP/IP を使用していない場合は、OSAX 装置で使用する VLAN を含むフェーズ IJBOCONF を使用します。

仮想 LAN を使用すると、物理ネットワークを管理上別々の論理ネットワークに分割できます。この論理ネットワークは、それぞれが物理的に独立しているように動作します。

z/VSE は、OSA Express (CHPID タイプ OSD および OSX) および HiperSockets™ 装置の VLAN をサポートします。

- レイヤー 3 構成では、IPv6/VSE および TCP/IP for z/VSE で VLAN を透過的に使用できます。
- IPv6 トラフィックを使用するレイヤー 2 構成で OSA-Express (CHPID タイプ OSD および OSX) 装置用に VLAN を構成するには、IPv6/VSE が必要です。

グローバル VLAN サポート:

- 1 つのリンクにつき 1 つのグローバル VLAN を定義できます。
- グローバル VLAN を定義した場合、同じリンクに対して他の VLAN を定義することはできません。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。本書は他の言語で IBM から提供されている場合があります。ただし、これを入手するには、本製品または当該言語版製品を所有している必要がある場合があります。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒*103-8510*
東京都中央区日本橋箱崎町*19*番*21*号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名前はすべて架空のものであり、類似する人物や企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

プログラミング・インターフェース情報

本書には、プログラムを作成するユーザーが z/VSE のサービスを使用するためのプログラミング・インターフェースが記述されています。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

製品資料のご使用条件

これらの資料は、以下のご使用条件に同意していただける場合に限りご使用いただけます。

適用される条件

IBM Web サイトの「ご利用条件」に加えて、以下のご使用条件が適用されます。

個人使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

権利

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

アクセシビリティー

アクセシビリティー機能は、運動障害または視覚障害など身体に障害を持つユーザーがソフトウェア製品を快適に使用できるようにサポートします。 z/VSE のアクセシビリティーの主要機能により、ユーザーは以下のことができるようになります。

- 画面読み上げ機能および画面拡大機能などの支援機能の使用
- キーボードのみを使用して、特定の機能または画面を使用したのと同等の機能を操作
- 色、コントラスト、フォント・サイズなど表示属性のカスタマイズ

支援機能の使用

画面読み上げ機能などの支援機能は、z/VSE のユーザー・インターフェースを使用して機能します。こうした支援機能を使用して z/VSE インターフェースにアクセスする場合、各機能固有の情報については支援機能の資料を参照してください。

資料の形式

本製品の資料は、Adobe Portable Document Format (PDF) で提供され、アクセシビリティー標準に準拠しています。PDF ファイルの使用に問題があり、Web ベース形式の資料を必要とする場合は、s390id@de.ibm.com 宛てに E メールを送信するか、または下記の宛先まで書面でご請求ください。

```
IBM Deutschland Research & Development GmbH
Department 3282
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany
```

この請求には必ず、資料番号および表題を付記してください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

用語集

この用語集には、IBM z/VSE の用語および定義が含まれています。

この用語集では、以下の相互参照を使用します。
1. 「を参照」は、ある用語から推奨される同義語への参照、あるいは頭字語または省略形から定義済みの完全な形式への参照を示します。
2. 「も参照」は、関連のある用語または対比する用語への参照を示します。

他の IBM 製品の用語集を参照するには、www.ibm.com/software/globalization/terminology にアクセスしてください。

A

アクセス制御ロギング報告機能 **(Access Control Logging and Reporting).** 保護データへのアクセスの試行をすべてログに記録し、そのような試行に関する報告書を選択した形式で印刷する IBM ライセンス・プログラム。

アクセス制御テーブル **(access control table (DTSECTAB)).** ユーザーが所定のリソースにアクセスするための権利を検査するために、システムで使用されるテーブル。

アクセス・リスト **(access list).** プログラムが参照できるアドレス・スペースまたはデータ・スペースを各項目が指定する表。

アクセス方式 **(access method).** ファイルまたはアドレスの定義およびそれらの間のデータ移動を行うためのプログラム。すなわち一組のコマンド (マクロ)。VSE/VSAM や VTAM がこの例。

アカウント・ファイル **(account file).** VSE/POWER によって維持されるディスク・ファイル。VSE/POWER および VSE/POWER で実行されるプログラムによって生成されたアカウンティング情報が含まれている。

アドレッシング・モード **(AMODE) (addressing mode (AMODE)).** プログラムに制御が渡されたときに処理できるアドレス長を指すプログラム属性。アドレスの長さは、24 ビット、31 ビットまたは 64 ビットのいずれか。24 ビット・アドレッシング・モードでは、プロセッサーはすべての仮想アドレスを 24 ビット値として扱う。31 ビット・アドレッシング・モードでは、プロセッサーはすべての仮想アドレスを 31 ビット値として扱う。64 ビット・アドレッシング・モードでは、プロセッサーはすべての仮想アドレスを 64 ビット値として扱う。ANY をアドレッシング・モードとするプログラムは、24 ビット・アドレッシング・モードまたは 31 ビット・アドレッシング・モードのいずれでも制御を受け取ることができる。64 ビット・アドレッシング・モードは、プログラム属性として使用できない。

管理コンソール **(administration console).** z/VSE において、すべてのシステム・メッセージを受信する 1 つ以上のコンソール。ただし、特定のコンソールに対するメッセージは除く。特定のコンソールに対するメッセージ (例えば、メッセージをそのコンソールにエコー出力する要求とともに実行依頼されたジョブから出されたメッセージ) のみを受信する「ユーザー・コンソール (user console)」と対比。管理コンソールのオペレーターは、すべての未処理のメッセージに応答して、すべてのシステム・コマンドを入力できる。

代替ブロック **(alternate block).** FBA ディスク上で、欠陥ブロックの代わりにデータを格納するように指定されているブロック。

代替索引 **(alternate index).** VSE/VSAM を使用するシステムにおいて、代替キー (つまり基本クラスターの基本キー以外のキー) に基づいて編成した基本クラスターの索引項目。例えば、仮に名前で順序付けされた人事ファイルは、部門番号でも索引付けできる。

代替ライブラリー **(alternate library).** ある端末のユーザーがライブラリーへの接続要求またはライブラリーの切り替え要求を出したときにその端末からアクセスできる、対話式にアクセス可能なライブラリー。

代替トラック **(alternate track).** ある端末のユーザーが (ライブラリーの) 接続要求または切り替え要求を出したときにその端末からアクセス可能になるライブラリー。

AMODE. アドレッシング・モード (addressing mode)。

APA. 全点アドレス可能。

APAR. プログラム診断依頼書。

付加ルーチン **(appendage routine).** 物理的にプログラムまたはサブシステム内にあるが、論理的には監視プログラム・ルーチンの拡張であるコードの部分。

アプリケーション・プロファイル **(application profile).** 1 つまたは複数のアプリケーション・プログラムの特性が格納されている制御ブロック。

アプリケーション・プログラム **(application program).** ユーザーのために、またはユーザーによって作成されたプログラムであり、ユーザーの作業に直接に使用される。在庫管理や給与計算のプログラムがこの例。「バッチ・プログラム (batch program)」および「オンライン・アプリケーション・プログラム (online application program)」も参照。

AR/GPR. アクセス・レジスターと汎用レジスターのペア。

ASC モード **(ASC mode).** アドレス・スペース制御モード。

ASI (自動化システム初期設定**)** プロシージャー **(ASI (automated system initialization) procedure).** 自動化システム初期設定の値を指定する一連の制御ステートメント。

アテンション・ルーチン **(attention routine (AR)).** オペレーターがアテンション・キーを押したときに制御を受け取るシステム・ルーチン。このルーチンは、コマンド入力にむけたコンソールのセットアップ、コマンドの読み取り、およびコマンドが要求したシステム・サービスの開始を実行する。

自動化システム初期設定 **(ASI) (automated system initialization (ASI)).** システム始動のための制御情報をカタログに登録し、システム始動時に自動的に取り出されるようにする機能。

自動開始 **(autostart).** オペレーターによる最小限の介入で、またはオペレーターによる介入なしで VSE/POWER を始動する機能。

補助ストレージ **(auxiliary storage).** プロセッサーの一部ではないアドレス可能ストレージ。例えば、ディスク装置上のストレージ。「外部ストレージ (external storage)」と同じ。

B

B 一時 **(B-transient).** 先頭に $$B を使用する名前を持ち、論理一時域 (LTA) 内で実行されるフェーズ。そのようなフェーズは、特殊な監視プログラム呼び出しによって活動化される。

境界 **(bar).** 2 ギガバイト (GB) 境界。

基本通信アクセス方式 **(BTAM) (basic telecommunications access method (BTAM)).** リモート装置との読み取り通信および書き込み通信を許可するアクセス方式。BTAM は z/VSE ではサポートされない。

BIG-DASD. 大容量 DASD のサブタイプ。64 K トラックを超える容量があり、ディスクの 10017 シリンダーまでを使用する。

ブロック **(block).** 通常 1 ブロックは 1 つのファイルの複数のレコードから成り、1 単位として伝送されるもの。しかし、レコードが非常に長い場合は、1 ブロックを 1 レコードの一部だけとすることもできる。FBA ディスクでは、1 ブロックはデータの 512 バイトのストリングである。「制御ブロック (control block)」も参照。

ブロック・グループ **(block group).** VSE/POWER において、固定ブロック方式 (FBA) 装置の基本的な組織単位。それぞれのブロック・グループは、数個の「転送単位」すなわちブロックで構成される。

C

CA 分割 **(CA splitting).** VSE JavaBeans のホスト部分であり、z/VSE のインストール時に読み取りキューに入れられるジョブ STARTVCS を使用して開始される。デフォルトで動的クラス R で実行される。VSE/VSAM において、指定されたフリー・スペースの最小量が追加のデータによって使い果たされたときに、制御域を動的に 2 倍にして、その CI を均等に分散するためのもの。

紙送り制御文字 **(carriage control character).** 印刷される出力レコード (行) の先頭文字。次の行が印刷される前にスキップする必要がある行数を判別する。

カタログ **(catalog).** ファイルおよびライブラリーのディレクトリーで、それらの位置を示すもの。カタログには、その他の情報 (ファイルを格納する装置のタイプ、パスワード、ブロック化因数など) が含まれることもある。サブライブラリー内にフェーズ、モジュール、またはブックなどのライブラリー・メンバーを保管するためのもの。「VSE/VSAM カタログ (VSE/VSAM catalog)」も参照。

セル・プール **(cell pool).** アプリケーション・プログラムから得られる仮想記憶域であり、呼び出し可能セル・プール・サービスによって管理される。セル・プールはアドレス・スペースまたはデータ・スペースにあり、最低 1 つのエクステントを持つアンカー域を含み、また同じサイズのセルを複数含んでいる。

中央設置場所 **(central location).** コンピューター・システムの制御装置 (通常は、コンピューター室内のシステム・コンソール) が設置されている場所。

チェーン・サブライブラリー **(chained sublibraries).** サブライブラリーで特定のライブラリー・メンバーを検索する順序を指定することにより、サブライブラリーをチェーニングできるようにする機能。

チェーニング **(chaining).** 同じタイプのメンバー (例えば、フェーズまたはオブジェクト・モジュール) をシステムが検索する、サブライブラリーの論理接続。

チャネル・コマンド・ワード **(CCW) (channel command word (CCW)).** チャネル・アドレス・ワードで指定された主記憶域内の場所にあるダブルワード。1 つ以上の CCW が、データ・チャネルの動作を指示するチャネル・プログラムを構成する。

チャネル・プログラム **(channel program).** 一連のデータ・チャネル操作を制御する 1 つまたは複数のチャネル・コマンド・ワード。この順序の実行は、サブチャネル開始命令によって開始される。

チャネル・スケジューラー **(channel scheduler).** 監視プログラムの中で、すべての入出力操作を制御する部分。

チャネル・サブシステム **(channel subsystem).** IBM Z に対して広範な追加チャネル (入出力) 機能を提供する z/Architecture の機能。

チャネル間接続 **(CTCA) (channel to channel attachment (CTCA)).** 以下の環境において、データを交換できるようにする機能。
1. VM で実行されている 2 台の仮想 VSE マシンの間で VSE/POWER の制御下で行う。または、
2. 2 台のプロセッサーの間で VTAM の制御下で行う。

文字コード化要求 **(character-coded request).** コード化され、文字ストリングとして転送される要求。「フィールド形式化要求 *(field-formatted request)*」 と対比。

チェックポイント **(checkpoint).**
1. ジョブ・ステップを後で再開できるように、ジョブおよびシステムの状況に関する情報を記録しておくことができるポイント。
2. そのような情報を記録するためのもの。

CICS (顧客情報管理システム**) (CICS (Customer Information Control System)).** 端末ユーザーとデータベースとの間のオンライン通信を制御する IBM プログラム。リモート端末で入力されたトランザクションは、ユーザー作成のアプリケーション・プログラムによって並行して処理される。プログラムには、データベースの構築、使用、および保守のための機能が含まれている。

CICS ECI. CICS 外部呼び出しインターフェース (ECI) は、CICS Transaction Server for z/VSE によって提供される *CICS* ビジネス論理インターフェース の 1 つの可能なリクエスター・タイプ。これは CICS クライアントの一部であり、z/VSE ホストで CICS 機能に対してワークステーション・プログラムを許可する。

CICS EXCI. 外部 CICS インターフェース (EXCI) は、CICS Transaction Server for z/VSE によって提供される *CICS* ビジネス論理インターフェース の 1 つの可能なリクエスター・タイプ。これは、すべての BSE バッチ・アプリケーションから CICS 機能を呼び出すことができるようにする。

CICS システム定義データ・セット **(CSD) (CICS system definition data set (CSD)).** オンライン・リソース定義 (RDO) を使用して CICS に定義されたすべてのレコードに対するリソース定義レコードを収めた VSAM KSDS クラスター。

CICS Transaction Server for z/VSE. 端末ユーザーとデータベースとの間のオンライン通信を制御する z/VSE の基本プログラム。これは、CICS/VSE の後継のシステムです。

CICS TS. CICS トランザクション・サーバー

CICS/VSE. 顧客情報管理システム (Customer Information Control System/VSE(CICS/VSE))。現在は拡張基本テープでは出荷されず、サポートも行われていない。z/VSE 5.1 以降では実行できない。

クラス **(class).** VSE/POWER において、同じ入力装置からの、または同じ出力装置へのジョブのグループ。

CMS. z/VM で実行される会話型モニター・システム。

共通ライブラリー **(common library).** ライブラリーを所有しているシステム (サブシステム) の任意のユーザーが対話式でアクセスできるライブラリー。

通信アダプター **(communication adapter).** 関連ソフトウェアが実装された回路カード。このカードを通して、プロセッサー、コントローラー、またはその他の装置をネットワークに接続することができる。

連絡領域 **(communication region).** プログラム内およびプログラム間での情報の転送のために確保しておく監視プログラムの領域。

コンポーネント **(component).**

1. コンピューター・システムの一部であるハードウェアまたはソフトウェア。
2. コンポーネント ID によって識別される、製品の機能部分。
3. z/VSE では、VSE/POWER または VTAM などのコンポーネント・プログラム。
4. VSE/VSAM では、格納されたレコードの名前付きのカタログされたグループ。例えば、キー順ファイルのデータ・コンポーネントまたは索引コンポーネント、または代替索引。

コンポーネント **ID (component identifier).** コンポーネントを MSHP に対して一意的に定義する 12 バイトの英数字ストリング。

条件付きジョブ制御 **(conditional job control).** ジョブ制御プログラムにおいて、このプログラムがテストする条件に基づいて 1 つまたは複数のステートメントを処理またはスキップする機能。

接続 **(connect).** 最下位レベルでライブラリー・アクセスを許可するためのもの。特定のサブライブラリーを使用するには、「読み取り」または「書き込み」などの修飾子が必要である。

接続プーリング **(connection pooling).** CICS TS で z/VSE データベース・コネクターの接続を管理 (再利用) するために、z/VSE 5.1 の更新で導入された。

コネクター **(connector).** z/VSE のコンテキストでは、コネクターは、2 つのプラットフォーム (Web クライアントと z/VSE ホスト、中間層と z/VSE ホスト、または Web クライアントと中間層) を接続するためのミドルウェアを提供する。

コネクター **(e-business** コネクター**) (connector (e-business connector)).** 異機種混合環境に接続するために提供されるソフトウェアの部分。大部分のコネクターが、z/VSE 以外の Java 対応プラットフォームと通信する。

コンテナー **(container).** IBM WebSphere Application Server などのアプリケーション・サーバーの JVM の一部であり、リソース管理およびトランザクション管理のリソースを提供することによって、サーブレット、EJB、および JSP の実装を容易にする。例えば、EJB 開発者は、アプリケーション・サーバーの JVM に対してコーディングできないが、コンテナーによって提供されるインターフェースに対してはコーディングできる。コンテナーの主な役割は、EJB とクライアントの間の中継として機能することである。これは VSE JavaBeans のホスト部分であり、z/VSE のインストール時に読み取りキューに入れられるジョブ STARTVCS を使用して開始される。デフォルトで動的クラス R で実行され、複数の EJB インスタンスを管理するためのものである。作成された EJB は、アプリケーション・サーバー上にあるコンテナーに保管する必要がある。コンテナーはその後、すべてのスレッド化、および EJB とのクライアント対話を管理し、接続プーリングおよびインスタンス・プーリングを調整する。

制御インターバル **(CI) (control interval (CI)).** VSE/VSAM がレコードを保管し、フリー・スペースを分散化するディスク・ストレージの固定長域。これは、VSE/VSAM がディスク・ストレージとの間で受け渡しする情報の単位である。FBA の場合、クラスター定義では、ブロック・サイズの整数倍で制御インターバルを定義する必要がある。

制御プログラム **(control program).** システムにおいて、プログラムの実行をスケジュールし、監視するためのプログラム。

会話型モニター・システム **(CMS) (conversational monitor system (CMS)).** 仮想計算機オペレーティング・システムであり、一般的な対話式タイム・シェアリング機能、問題解決機能、およびプログラム開発機能を提供し、z/VM の制御下で作動する。

カウント・キー・データ **(CKD)** 装置 **(count-key-data (CKD) device).** データを次のようなレコード形式で保管するディスク装置。すなわち、カウント・フィールド、キー・フィールド、データ・フィールド。カウント・フィールドには、他の情報と一緒にレコードのアドレスが特定の形式で含まれている。その形式は、シリンダー番号、ヘッド (トラック) 番号、レコード番号、およびデータ・フィールドの長さである。キー・フィールド (存在する場合) には、レコードのキーまたは検索引数が含まれている。CKD ディスク・スペースは、トラックおよびシリンダー単位で割り振られる。「*FBA* ディスク装置 *(FBA disk device)*」 と対比。「拡張 CKD 装置 (extended count-key-data device)」も参照。

区画間連絡制御 **(cross-partition communication control).** VSE サブシステムとユーザー・プログラムが相互に通信できるようにする機能。例えば、VSE/POWER。

暗号トークン **(cryptographic token).** 通常は単にトークン と呼ばれ、デジタル署名の生成またはデータの暗号化などの暗号機能を実行するためのインターフェースを提供する装置。

暗号化 **(cryptography).**
1. 情報を暗号文と呼ばれる読めない形式に変換 (暗号化) して、その情報を保護する方法。秘密鍵を持っているユーザーのみがメッセージを平文に復号 (暗号化解除) できる。

2. 情報の内容を隠すためにデータを変換し、無許可の使用や検出されない変更を防止すること。

D

データ・ブロック・グループ **(data block group).** データ・ファイル上で VSE/POWER ジョブに割り振ることができるスペースの最小単位。この割り振りは装置特性には関係しない。

データ変換記述子ファイル **(DCDF) (data conversion descriptor file (DCDF)).** DCDF を使用すると、PC とそのホストの間でデータを転送する際に、レコード内の個々のフィールドを変換できる。DCDF は、PC 環境とホスト環境の両方に対して特定のファイルのレコード・フィールドを定義する。

データのインポート **(data import).** あるオペレーティング・システムで使用していたデータを、継続して別のオペレーティング・システムで使用できるように再形式設定するプロセス。

ファイル間データ転送、テスト、および操作 **(DITTO)** ユーティリティー **(Data Interfile Transfer, Testing, and Operations (DITTO) utility).** カード入出力装置、テープ装置、およびディスク装置用のファイル間サービスを提供する IBM プログラム。最新バージョンは DITTO/ESA for VSE。

データ言語 **/I (Data Language/I (DL/I)).** CICS で使用されるデータベース・アクセス言語。

データ・リンク **(data link).** SNA において、リンク接続と、ネットワーク・ノードを結合するリンク・ステーションとの組み合わせ。例えば、z/Architecture チャネルとこれに関連付けられたプロトコルとの組み合わせ。論理リンクと物理リンクの両方がある。

データ・セキュリティー **(data security).** 偶発的か意図的かを問わず、無許可の開示、転送、変更または破壊に対してデータを保護する。

データ・セット・ヘッダー・レコード **(data set header record).** VSE/POWER では、DSHR と略される。別名は NDH または DSH。出力データの前または入力データの中間のいずれかにある NJE 制御レコードで、データ形式の変更を示す。

データ・スペース **(data space).** z/Architecture の命令を通じてプログラムが直接操作できる、最大 2 ギガバイトの連続仮想記憶域アドレスの範囲。アドレス・スペースとは異なり、データ・スペースはユーザー・データのみを保持できる。これには共有域、プログラムはいずれも含まれない。データ・スペースでは命令は実行しない。「アドレス・スペース (address space)」と対比。

データ端末装置 **(DTE) (data terminal equipment (DTE)).** SNA において、データ送信側、データ受信側、またはその両方として機能するデータ装置の一部。

データベース・コネクター **(database connector).** z/VSE 5.1.1 で導入された機能であり、クライアント部分とサーバー部分からなる。クライアントは、z/VSE でアプリケーションによって使用される API (CBCLI) を提供し、Java 対応プラットフォーム上のサーバーは、データベースによって提供される JDBC ドライバーを接続する。クライアントとサーバーの両方が TCP/IP を介して通信する。

Database 2 (Db2). IBM のリレーショナル・データベース管理システム。

Db2 ベース・コネクター **(Db2-based connector).** VSE/ESA 2.5 で導入された機能であり、VSAM および DL/I 機能とともに、Db2 ストアード・プロシージャーを使用して Db2、VSAM、および DL/I のデータにアクセスできるようにするカスタマイズ済みの Db2 バージョンが含まれている。

Db2 Runtime only Client Edition. Client Edition for z/VSE には、z/VSE および Linux on z Systems を統合するために、いくつかの拡張機能が付属しており、パフォーマンスが改善されている。

Db2 ストアード・プロシージャー **(Db2 Stored Procedure).** z/VSE のコンテキストでは、Db2 ストアード・プロシージャーは、Db2 データにアクセスする Language Environment (LE) プログラムである。ただし、VSE/ESA 2.5 以降では、Db2 ストアード・プロシージャーを使用して VSAM データおよび DL/I データにアクセスすることもできる。このようにして、VSAM と Db2 の間でデータを交換できる。

DBLK. データ・ブロック (Data block)。

DCDF. データ変換記述子ファイル。

非ブロック化 **(deblocking).** ブロックの各レコードを処理できるようにするプロセス。

専用 **(**ディスク**)** 装置 **(dedicated (disk) device).** 複数のユーザーで共有することができないディスク装置。

装置アドレス **(device address).**
1. 入出力装置をその装置番号で識別するもの。
2. データ通信においては、データが送信可能または受信可能な装置の識別。

装置駆動システム **(DDS) (device driving system (DDS)).** CICS スプーラーまたは PSF など、VSE/POWER 外にあるソフトウェア・システム。宛先装置にスプール出力を書き込む。

装置サポート機能 **(DSF) (Device Support Facilities (DSF)).** IBM プログラムおよびユーザー・プログラムからディスク・ボリュームにアクセスできるようにするため、ディスク・ボリューム上で操作を実行する IBM 提供のシステム制御プログラム。これらの操作の例としては、ディスク・ボリュームの初期設定および代替トラックの割り当てがある。

装置タイプ・コード **(device type code).** 4 桁または 5 桁のコードであり、コンピューター・システムに対して入出力装置を定義するために使用される。 **ICKDSF** も参照。

ダイアログ **(dialog).** 対話式システムでは、一連の関連した照会と応答のことで、2 人の人間の間で行われる会話と類似している。z/VSE では、特定の作業 (例えば、ファイルの定義など) を行うために使用する一連のパネル。

ダイアログ・マネージャー **(dialog manager).** ユーザーとシステムの間の通信を容易にする z/VSE のプログラム・コンポーネント。

デジタル署名 **(digital signature).** コンピューター・セキュリティーにおいて、受信側が送信側の ID を証明できるようにする、メッセージまたはメッセージの一部に付加された暗号化されたデータ。

デジタル署名アルゴリズム **(DSA) (Digital Signature Algorithm (DSA)).** デジタル署名アルゴリズムは、米国政府によって定義されたデジタル署名用の規格。DSA デジタル署名は、1 組の規則 (すなわち DSA) と、署名者の ID およびデータの保全性を検証できるような一連のパラメーターを使用して計算された大きな数のペアである。DSA は、署名を生成して検証する機能を提供する。

ディレクトリー **(directory).** z/VSE において、プログラム・ライブラリーの索引。

直接アクセス **(direct access).** ストレージ・デバイス上のデータに、その順序ではなくアドレスを使用してアクセスすること。磁気テープの場合とは逆にディスク装置ではこれが典型的なアクセスである。「順次アクセス *(sequential access)*」と対比。

ディスク・オペレーティング・システム常駐ボリューム **(DOSRES) (disk operating system residence volume (DOSRES)).** システム始動に必要なプログラムおよびプロシージャーが入っている、システム・サブライブラリー IJSYSRS.SYSLIB があるディスク・ボリューム。

ディスク共用 **(disk sharing).** 独立した複数のコンピューター・システムで、共用ディスク装置上の共通データを使用できるようにするオプション。

後処置 **(disposition).** ジョブ入力項目またはジョブ出力項目の処理方法を VSE/POWER に指示する手段。RDR/LST/PUN キューにある項目は項目のローカル後処理に従って処理され、XMT キューにある項目は項目の伝送後処理に従って処理される。例えば、ジョブは処理後に削除されるか保持される。

配布テープ **(distribution tape).** z/VSE のような事前構成オペレーティング・システムを含む磁気テープ。このテープは、プログラム・インストールのために、ユーザーに提供される。

DITTO/ESA (VSE 版) (DITTO/ESA for VSE). ファイル間データ転送、テスト、および操作ユーティリティー。ディスク装置、テープ装置、およびカード装置用のファイル間サービスを提供する IBM プログラム。

DSF. 装置サポート機能。

DSH (R). データ・セット・ヘッダー・レコード。

ダミー装置 **(dummy device).** 実在の入出力装置に関連付けられていない装置アドレス。この装置アドレスにおける入出力は、ディスク上でスプールされる。

二重 **(duplex).** 同時に送受信できるデータ通信に関連した用語。

DU-AL (ディスパッチ可能単位 - アクセス・リスト) (DU-AL (dispatchable unit - access list)). z/VSE メインタスクまたはサブタスクに関連付けられたアクセス・リスト。プログラムは、そのタスクに関連付けられた DU-AL、およびその区画に関連付けられた PASN-AL を使用する。「*PASN-AL*」 も参照。

動的クラス・テーブル **(dynamic class table).** 動的区画の特性を定義するテーブル。

動的区画 **(dynamic partition).** 固定された静的割り振りを使用せずに、「必要に応じて」作成および活動化される区画。処理後、占有されたスペースは解放される。動的区画はクラスによってグループ化され、ジョブはクラスによってスケジュールされる。「静的区画 *(static partition)*」と対比。

動的スペース・レクラメーション **(dynamic space reclamation).** 自動的に再利用可能になるように、ライブラリー・メンバーの削除によって解放されるスペースに対して提供されるライブラリアン機能。

E

ECI. 「*CICS ECI*」 を参照。

エミュレーション **(emulation).** プログラミング手法および特別のコンピューター機能を使用して、別のシステム用または使用可能な装置とは異なる入出力装置を使用するように書かれたプログラムを、コンピューター・システムで実行できるようにすること。

エミュレーション・プログラム **(EP) (emulation program (EP)).** IBM の制御プログラムであり、チャネル接続された 3705 または 3725 の通信コントローラーが、IBM 2701 データ・アダプター装置または IBM 2703 伝送制御をエミュレートできるようにする。

エンド・ユーザー **(end user).**
1. アプリケーション・プログラムを使用する人。
2. SNA において、SNA ネットワークを通るユーザー・データの最終的な送信元または宛先。アプリケーション・プログラムまたは端末オペレーターが考えられる。

Enterprise Java Bean. EJB は分散 Bean である。「分散」とは、EJB の一部分が Web アプリケーション・サーバーの JVM 内で実行されるのに対して、他の部分は Web ブラウザーの JVM 内で実行されることを意味する。EJB は、データベース内の 1 つのデータ行 (エンティティー Bean)、またはリモート・データベースへの 1 つの接続 (セッション Bean) のいずれかを表す。通常、両方のタイプの EJB が一緒に作動する。これによって、リレーショナル・データおよび非リレーショナル・データが混在する異機種混合環境において、標準化された方法でデータを表しデータにアクセスすることが可能になる。「*JavaBean*」 も参照。

入力順ファイル **(entry-sequenced file).** VSE/VSAM ファイルの 1 つで、レコードが内容に関係なくロードされ、レコードの相対バイト・アドレスが変らないデータ・セット。アドレス指定によるアクセスによりレコードの取得と格納が行われ、新しいレコードはファイルの終わりに追加される。

環境記録・編集・印刷 **(EREP)** プログラム **(Environmental Record Editing and Printing (EREP) program).** システム・レコード・ファイルに収められているデータを詳細分析のために使用できるようにする z/VSE 基本プログラム。

EPI. 「*CICS EPI*」 を参照。

ESCON チャネル **(**エンタープライズ・システム接続チャネル**) (ESCON Channel (Enterprise Systems Connection Channel)).** 光ファイバー・ケーブルを使うシリアル・チャネルであり、ホストと入出力装置用との間の高速接続を可能にする。これは、z114 まで ESA/390 および IBM Z 入出力インターフェースに従っている。zEC12 プロセッサーでは ESCON チャネルはサポートされない。

出口ルーチン **(exit routine).**
1. 2 つのルーチン・タイプ (インストール・システム出口ルーチンまたはユーザー出口ルーチン) のいずれか。「出口プログラム (exit program)」と同義。
2. 「ユーザー出口ルーチン *(user exit routine)*」を参照。

拡張アドレッシング・サポート **(extended addressability).** アドレス・スペース内または外にある、31 ビットまたは 64 ビットの仮想記憶域をプログラムが使用できる能力。

拡張リカバリー機能 **(XRF) (extended recovery facility (XRF)).** z/VSE における CICS の機能の 1 つで、ある CICS システムを他の CICS システムのバックアップとして使用することによって、CICS の可用性を強化するもの。

外部セキュリティー・マネージャー **(ESM) (External Security Manager (ESM)).** z/VSE の一部である基本セキュリティー・マネージャー (BSM) の場合と比較して、拡張された機能および柔軟性を提供することができる、有料のベンダー製品。

F

FASTCOPY. 「*VSE/*高速コピー・データ・セット・プログラム *(VSE/Fast Copy)*」を参照。

高速コピー・データ・セット・プログラム **(VSE/**高速コピー**) (fast copy data set program (VSE/Fast Copy)).** 「*VSE/*高速コピー・データ・セット・プログラム *(VSE/Fast Copy)*」を参照。

高速サービス・アップグレード **(Fast Service Upgrade (FSU)).** z/VSE のサービス機能で、リフレッシュ・リリースをライブラリー制御テーブルなどの制御情報を生成し直さないでインストールするためのもの。

FAT-DASD. 大容量 DASD のサブタイプ。4369 シリンダー (64 K トラック) を超える最大 64 K シリンダーまでの装置をサポートする。

FCOPY. 「*VSE*/高速コピー・データ・セット・プログラム *(VSE/Fast Copy)*」を参照。

フェンス **(fence).** プロセッサー複合体の 1 つ以上のコンポーネントまたはエレメントを、残りのコンポーネントまたはエレメントと分離すること。この分離は論理境界によって行われる。これによって、ユーザー操作と保守手順を同時に行うことができる。

取り出し **(fetch).**
1. 一定量のデータを見つけて、それをストレージからロードすること。
2. あるプログラム・フェーズをサブライブラリーから仮想記憶域に移し、そのフェーズに制御を渡すこと。
3. 2 を実行するために使用されるマクロ命令 (FETCH) の名前。「ローダー *(loader)*」 も参照。

ファイバー・チャネル・プロトコル **(FCP) (Fibre Channel Protocol (FCP)).** ファイバー・チャネル規格に準拠し、IBM zSeries プロセッサー上で FICON および FICON Express 機能カードを介してシステムと周辺装置の接続を可能にする、ハードウェアとソフトウェアの組み合わせ。z/VSE では、zSeries FCP は業界標準の SCSI ディスク装置にアクセスするために使用されます。

フラグメント化 **(**ストレージの**) (fragmentation (of storage)).** 仮想記憶域の実アドレス範囲または仮想アドレス範囲で、ストレージの未使用セクション (フラグメント) を割り振ることができないこと。

FSU. 高速サービス・アップグレード (Fast service upgrade)。

FULIST (機能リスト **; FUnction LIST).** ユーザーの選択用に一組のファイルまたは機能、あるいはその両方を表示する選択パネルの種類。

G

生成 **(generation).** 「マクロ生成 *(macro generation)*」を参照。

生成機能 **(generation feature).** プログラムのオブジェクト・コードをユーザーの要件に合わせて調整するために使用される IBM ライセンス・プログラムの注文オプション。

GETVIS スペース **(GETVIS space).** プログラムへの動的割り振りに使用できる、区画内または共有仮想記憶域内のストレージ・スペース。

ゲスト・システム **(guest system).** 別の (ホスト) システムの制御下で実行されるデータ処理システム。メインフレームでは、z/VSE は z/VM のゲストとして実行できる。

H

ハードウェア・ウェイト **(hard wait).** すべての操作が延期されたときの、プロセッサーの状態。ハードウェア・ウェイト状態からシステムをリカバリーするには、新規システムのスタートアップを実行させる必要がある。

ハッシュ関数 **(hash function).** ハッシュ関数は、可変サイズの入力データを受けて、ハッシュ値と呼ばれる固定サイズのストリングを返す変換である。暗号化では、ハッシュ関数には以下の追加のプロパティーがある。
- ハッシュ関数は計算が簡単。
- ハッシュ関数は片方向。つまり、「逆」関数を計算することは不可能。
- ハッシュ関数には衝突がない。つまり、異なる入力が同じハッシュ値になることは不可能。

ハッシュ値 **(hash value).** テキストにハッシュ関数 を適用した結果として得られる固定サイズのストリング。

高水準アセンブラー **(VSE 版) (High-Level Assembler for VSE).** 拡張アセンブラー・プログラミング・サポートを提供する、プログラミング言語。z/VSE の基本プログラム。

ホーム・インターフェース **(home interface).** 新規 EJB オブジェクトのインスタンス化、EJB のイントロスペクト、および EJB インスタンス化の削除を行うメソッドを提供する。リモート・インターフェースについては、デプロイメント・ツールによって実装クラスが生成されるため、これが必要になる。すべてのセッション Bean のホーム・インターフェースが、少なくとも 1 つの *create()* メソッドを提供する必要がある。

ホスト・モード **(host mode).** この動作モードでは、PC は VSE ホストをアクセスすることができ、プログラマブル・ワークステーション (PWS) 機能では、VSE の移動ユーティリティーを使用できる。

ホスト・システム **(host system).** データ通信構成において、制御または最高位のシステム。

ホスト転送ファイル **(HTF) (host transfer file (HTF)).** IBM パーソナル・コンピューターに送られたりそこから出されたりするファイルのための中間のストレージ域として、z/VSE のワークステーション・ファイル転送サポートにより使用される。

HTTP セッション (HTTP Session). z/VSE のコンテキストでは、サーブレットを呼び出す Web ブラウザー・クライアントを識別する (つまり、クライアントと中間層プラットフォームとの間の接続を識別する)。

I

ICCF. *VSE/ICCF* を参照してください。

ICKDSF (装置サポート機能) (Device Support Facilities). IBM ディスク装置のインストール、使用および保守をサポートする z/VSE 基本プログラム。

組み込み機能 **(include function).** プログラム入力に組み込むライブラリー・メンバーを取得する。

索引 **(index).**
1. 索引順次データ・セットまたは索引付きファイル内でレコードを見つけるために使用されるテーブル。
2. それぞれがキーとポインターで構成されるペアの順序付き集合であり、キー順データ・セットまたはキー順ファイルのレコードを順序付けて見つけるために使用される。索引レコードのいくつかのレベルで編成される。「代替索引 *(alternate index)*」も参照。

入出力制御システム **(IOCS) (input/output control system (IOCS)).** 主記憶装置と補助記憶装置との間のデータ転送を処理する一群の IBM 提供ルーチン。

通信統合アダプター **(ICA) (integrated communication adapter (ICA)).** 複数回線を接続できるプロセッサーの部分。

統合コンソール **(integrated console).** z/VSE において、z/VSE システム・コンソールとして動作する、IBM Z で使用可能なサービス・プロセッサー・コンソール。統合コンソールは、一般に、IPL 中に使用され、他のコンソールが使用できないときにリカバリーの目的で使用される。

対話計算・制御機能 **(ICCF) (Interactive Computing and Control Facility (ICCF)).** システムのプロセッサーにリンクされた端末の許可ユーザーに対して、タイム・スライス・ベースでインターフェースの役割を果たす IBM ライセンス・プログラム。

対話区画 **(interactive partition).** VSE/ICCF を経由して対話式にサブミットされたジョブ処理を目的とした仮想記憶。

対話式ユーザー通信ビークル **(IUCV) (Interactive User Communication Vehicle (IUCV)).** z/VM 下の操作用の VSE 監視プログラムにおいて使用可能なプログラミング・サポート。ユーザーはこのサポートを使用すると、優先されていないゲストと通信する場合と同じ方法で、他のユーザーまたは CP と通信することができる。

中間ストレージ **(intermediate storage).** データを処理する前に、そのデータを一時的に保持するために使用される任意の記憶装置。

IOCS. 入出力制御システム (Input/output control system)。

IPL. 初期プログラム・ロード (Initial program load)

リカバリー不能エラー **(irrecoverable error).** コンピューター・プログラムまたは実行の範囲外にあるリカバリー手法を使用しないとリカバリーできないエラー。

IUCV. 対話式ユーザー通信ビークル (Interactive User Communication Vehicle)。

J

JAR. プラットフォームに依存しないファイル形式で、多くのファイルを 1 つのファイルに集約する。複数のアプレットおよびその必要条件コンポーネント (.class ファイル、イメージ、および音) を 1 つの JAR ファイルにバンドルしてから、単一の HTTP トランザクションを使用して Web ブラウザーにダウンロードできる (ダウンロード速度が大幅に向上する)。また、JAR 形式では圧縮がサポートされ、ファイル・サイズが削減される (さらにダウンロード速度が向上する)。使用される圧縮アルゴリズムには、ZIP アルゴリズムとの完全な互換性がある。アプレットの所有者は、JAR ファイル内の個々の項目にデジタルに署名し、その発信元を認証することもできる。

Java アプリケーション (Java application). Web ブラウザーの JVM 内で実行される Java プログラム。プログラムのコードは、ローカル・ハード・ディスク上または LAN 上にある。Java アプリケーションは、グラフィカル・インターフェースを使用する大規模プログラムの場合がある。Java アプリケーションは、すべてのローカル・リソースに無制限にアクセスできる。

Java バイトコード (Java bytecode). バイトコードは、Java ソース言語ステートメントが入っているファイルのコンパイル時に作成される。コンパイル済み Java コード (つまり「バイトコード」) は、実行準備ができているプログラム・モジュールまたはファイル (一度に 1 つの命令が実行されるようにコンピューター上で実行される) と似ている。ただし、バイトコード内の命令は、実際に *Java* 仮想マシン に対する命令である。命令を一度に 1 つずつ解釈する代わりに、バイトコードは、ジャストインタイム (JIT) コンパイラーを使用し

てオペレーティング・システムのプラットフォームごとに再コンパイルされる。通常、これにより Java プログラムをより高速に実行できる。バイトコードは、接尾部 **.CLASS** を持つバイナリー・ファイルに入っている。

Java サーブレット (Java servlet). 「サーブレット *(servlet)*」 を参照。

JHR. ジョブ・ヘッダー・レコード。

ジョブ・アカウンティング・インターフェース **(job accounting interface).** ジョブ・ステップごとにアカウンティング情報を累積する機能。システム・ユーザーへの課金、新規アプリケーションの計画、およびシステム操作のより効率的な監視のために使用される。

ジョブ・アカウンティング・テーブル **(job accounting table).** 監視プログラム内で、ユーザー用の会計情報が累算される区域。

ジョブ・カタログ **(job catalog).** それぞれの DLBL ステートメントでファイル名 IJSYSUC を指定することによって、ジョブのために使用可能になるカタログ。

ジョブ入力制御言語 **(job entry control language (JECL)).** プログラマーがジョブの処理方法を VSE/POWER に指示するために使用される制御言語。

ジョブ・ステップ **(job step).** 1 回の実行に必要な JCL ステートメントを備えた関連するプログラムのグループのうちの 1 つ。すべてのジョブ・ステップは、ジョブ全体を表す 1 つの JOB ステートメントに続いて指定される EXEC ステートメントにより、ジョブ・ストリーム内で識別される。

ジョブ・トレーラー・レコード **(JTR) (job trailer record (JTR)).** VSE/POWER パラメーター JTR。別名 NJT。入力キューまたは出力キュー内のジョブ項目を終了し、アカウンティング情報を提供する NJE 制御レコード。

K

キー **(key).** VSE/VSAM において、索引項目またはレコード自身を識別し順序付けるためにデータ・レコードの特定のフィールド (キー・フィールド) から取り出された 1 つまたは複数の文字。

キー順 **(key sequence).** レコード自体または索引内にあるレコードのキーのいずれか、またはその両方の照合シーケンス。キー・シーケンスは英数字順になる。

キー順ファイル **(key-sequenced file).** レコードがキー順にロードされ、索引により制御される VSE/VSAM ファイル。レコードの検索および格納はキーに基づくアクセスまたはアドレスに基づくアクセスによって行われ、新しいレコードはキー順にファイルへ挿入される。

KSDS. キー順データ・セット。「キー順ファイル *(key-sequenced file)*」を参照。

L

ラベル **(label).**
1. テープ・ボリューム、ディスク・ボリューム、ディスケット・ボリューム、またはそのようなボリュームに入っているファイルの識別レコード。
2. アセンブリー言語プログラミングにおいて、一般にブランチに使用される名前の付いた命令。

ラベル情報域 **(label information area).** ジョブ制御ステートメントまたはコマンドから読み取ったラベル情報を保管するためのディスク上の区域。「ラベル域 *(label area)*」と同じ。

Language Environment for z/VSE. VSE プラットフォーム上での Language Environment の実装となる IBM ソフトウェア・プロダクト。

言語翻訳プログラム **(language translator).** ある言語で書かれたステートメントを受け取って、別の言語の同等のステートメントを生成するアセンブラー、コンパイラー、その他のルーチンを総称する用語。

大容量 **DASD (Large DASD).** 以下を満たす DASD 装置。
1. 64 K トラックを超える容量を持つ。かつ
2. カタログによって所有され VSE/ESA 2.6 より前に作成された、VSAM スペースを持たない。

LE/VSE. Language Environment for z/VSE の略語。

ライブラリアン **(librarian).** システム・ライブラリーと専用ライブラリーを保守、サービス、そして編成する一連のプログラム。

ライブラリー・ブロック **(library block).** サブライブラリーに保管されているデータのブロック。

ライブラリー登録簿 **(library directory).** システムがアクセスするライブラリーの特定のサブライブラリーを見つけられるようにする索引。

ライブラリー・メンバー **(library member).** サブライブラリーに保管して取得できる、データの最小単位。

行コマンド **(line commands).** VSE/ICCF において、画面上の各行の宣言を変更する特殊なコマンド。例えば、行の宣言のコピー、移動、または削除が可能。

リンケージ・エディター **(linkage editor).** 独立して変換された 1 つ以上のオブジェクト・モジュールから、または 1 つ以上の既存のフェーズ (実行可能コード) から、あるいはこれらの両方から、フェーズを作成するために使用されるプログラム。フェーズの作成時に、リンケージ・エディターは、入力として使用できるモジュールとフェーズの間の相互参照を解決する。プログラムは新たに作成されたフェーズをカタログ登録できる。

リンケージ・スタック **(linkage stack).** システムからプログラムに与えられる保護ストレージ域。ブランチ・スタックまたはスタッキング・プログラム呼び出しの際に状況情報が保存される。

リンク・ステーション **(link station).** SNA において、ノードから接続できリンクを制御できる、ハードウェアとソフトウェアの組み合わせ。

ローダー **(loader).** データまたはプログラムを主記憶域内に読み込むルーチン (一般にはコンピューター・プログラム)。「再配置ローダー *(relocating loader)*」 も参照。

ローカル共用リソース **(LSR) (local shared resources (LSR)).** VSE/VSAM のオプションであり、ファイル間で制御ブロックを共有するために 3 つの追加のマクロによって活動化される。

ロック・ファイル **(lock file).** VSE の共有ディスク環境において、共有システムが共有データへのアクセスを制御するために使用するディスク上のシステム・ファイル。

論理区画 **(logical partition).** LPAR モードで、システム制御プログラムの動作をサポートするために定義されるサーバー・ハードウェアのサブセット。

論理レコード **(logical record).** 通常 1 つの主題に関連しデータ管理機能によって 1 単位として処理されるユーザー・レコード。「物理レコード *(physical record)*」と対比。物理レコードはこれより大きいことも小さいこともある。

論理装置 **(LU) (logical unit (LU)).**
1. プログラミングにおいて、入出力装置アドレスを表すために使用される名前。物理装置 *(PU)*、システム・サービス制御点 *(SSCP)*、*1* 次論理装置 *(PLU)*、および *2* 次論理装置 *(SLU)*。
2. SNA において、ユーザーが以下の目的で SNA ネットワークにアクセスする際に使用するポート。
   a. 別のユーザーと通信するため。および
   b. SSCP の機能にアクセスするため。LU では、少なくとも 2 つのセッションをサポートできる。1 つは SSCP とのセッションであり、もう 1 つは別の LU とのセッションである。その他の LU とのセッションを多数サポートできることがある。

論理装置名 **(logical unit name).** プログラミングにおいて、入出力装置のアドレスを表すために使用される名前。

論理装置 **6.2 (logical unit 6.2).** 分散処理環境におけるプログラム間通信のための SNA/SDLC プロトコル。 LU 6.2 には、次のような特徴がある。
1. セッション・パートナーとは対等の関係にある。
2. 1 つのセッションを複数のトランザクションで効率よく利用できる。
3. 包括的なエンドツーエンドのエラー処理。
4. 製品の実装にマップされた構造化済み verb で構成される、汎用アプリケーション・プログラミング・インターフェース (API)。

ログオン解釈ルーチン **(logons interpret routine).** VTAM において、ログオン情報を変換し、解釈テーブル項目に関連付けられたインストール・システム出口ルーチン。ログオンの検査も行う。

LPAR モード **(LPAR mode).** 論理分割モード (Logically partitioned mode)。PR/SM 機構がインストールされているとき、構成 (CONFIG) フレーム上で使用可能となる CP モード。LPAR モードでは、オペレーターは、処理装置のハードウェア・リソースを複数の論理区画に割り振ることができる。

M

マクロ定義 **(macro definition).** 単一のソース・ステートメントから一連のアセンブラー・ステートメントと機械語命令を生成するための名前、形式、および条件を定義する一組のステートメントおよび命令。

マクロ展開 **(macro expansion).** 「マクロ生成 *(macro generation)*」 を参照。

マクロ生成 **(macro generation).** ステートメントの定義によってマクロ命令がプログラム内で置換される、アセンブラー操作。この操作は、アセンブリーの前に行われる。「マクロ展開 *(macro expansion)*」 と同じ。

マクロ **(命令) (macro (instruction)).**
1. アセンブラー・プログラミングにおいて、マクロ定義で既に定義されている一連のステートメントをアセンブラーに処理させる、ユーザーが作成したアセンブラー・ステートメント。

2. ある要求に対する応答として、特定のアクションがある順番で実行されるように定義された一連の VSE/ICCF コマンド。

システム・ヒストリー保守プログラム **(MSHP) (maintain system history program (MSHP)).** VSE システムに関する各種のインストール活動、調整活動、および保守活動を自動化したり、制御したりするために使用されるプログラム。

メインタスク **(main task).** マルチプログラミング環境内の区画にあるメインプログラム。

マスター・コンソール **(master console).** z/VSE において、すべてのシステム・メッセージを受信する 1 つ以上のコンソール。ただし、特定のコンソールに対するメッセージは除く。特定のコンソールに対するメッセージ (例えば、メッセージをそのコンソールにエコー出力する要求とともに実行依頼されたジョブから出されたメッセージ) のみを受信する「ユーザー・コンソール *(user console)*」 と対比。マスター・コンソールのオペレーターは、すべての未解決なメッセージに応答し、すべてのシステム・コマンドを入力することができる。

最大 **CA (maximum (max) CA).** カウント・キー・データまたは固定ブロック装置の最大制御域サイズに相当する割り振り単位。CKD 装置の場合、最大 CA は 1 シリンダーに相当する。

メモリー・オブジェクト **(memory object).** IARV64 マクロを使用して作成される、2 GB 境界より上に割り振られる仮想記憶のチャンク。

メッセージ **(message).** VSE では、プログラムからオペレーターまたはユーザーに送られる通知。これは、コンソール、ディスプレイ、または印刷出力で表示できる。

MSHP. 「システム・ヒストリー保守プログラム (maintain system history program)」を参照。

マルチタスキング **(multitasking).** 同一の区画で、1 つのメインタスクと 1 つ以上のサブタスクが並行して実行されること。

MVS. 多重仮想ストレージ。MVS/390、MVS/XA、MVS/ESA、および z/OS (OS/390) オペレーティング・システムの MVS エレメントを意味する。

N

NetView. ネットワークをモニターし、管理して、ネットワークの問題を診断するために使用される z/VSE のオプション・プログラム。

ネットワーク・アドレス **(network address).** SNA で、サブエリアとエレメント・フィールドから構成されるアドレスであり、リンク、リンク・ステーション、または NAU を識別する。サブエリア・ノードはネットワーク・アドレスを使用し、周辺ノードはローカル・アドレスを使用する。周辺ノードに接続されているサブエリア・ノードの境界機能は、ローカル・アドレスをネットワーク・アドレスに変換し、ネットワーク・アドレスをローカル・アドレスに変換する。「ネットワーク名 *(network name)*」も参照。

ネットワーク・アドレス可能装置 **(NAU) (network addressable unit (NAU)).** SNA では、論理装置、物理装置、またはシステム・サービス制御点。パス制御ネットワークによって伝送される情報の発信元または宛先。各 NAU には、パス制御ネットワークに対して NAU を表すネットワーク・アドレスがある。「ネットワーク名 *(network name)*」、「ネットワーク・アドレス *(network address)*」 も参照。

ネットワーク制御プログラム **(NCP) (Network Control Program (NCP)).** シングル・ドメイン、マルチドメイン、および相互接続ネットワーク機能のために、通信コントローラー・サポートを提供する IBM ライセンス・プログラム。フルネームは ACF/NCP。

ネットワーク定義テーブル **(NDT) (network definition table (NDT)).** VSE/POWER ネットワーキングにおいて、ネットワーク内のすべてのノードがリストされているテーブル。

ネットワーク名 **(network name).**
1. SNA において、ユーザーが NAU、リンク、またはリンク・ステーションを参照するために使用する記号 ID。「ネットワーク・アドレス *(network address)*」も参照。
2. マルチドメイン・ネットワークにおいて、VTAM アプリケーション・プログラムを定義する APPL ステートメントの名前。このネットワーク名は、ドメイン間で固有でなければならない。

ノード **(node).**
1. SNA において、ネットワーク内の複数のリンクに共通したリンクの終点または接合点。ノードは、ホスト・プロセッサー、通信コントローラー、クラスター・コントローラー、端末に分散させることができる。ルーティングやその他の機能がもつ能力は、ノードによって異なることが可能である。
2. VTAM において、記号名で定義されたネットワーク内の点。「ネットワーク・ノード *(network node)*」と同じ。「大ノード *(major node)*」および「小ノード *(minor node)*」を参照。

ノード・タイプ **(node type).** SNA において、サポートするプロトコル、および含むことができるネットワーク・アドレス式装置 (NAU) によって、ノードを指定すること。

O

オブジェクト・モジュール **(プログラム) (object module (program)).** アセンブラーまたはコンパイラーの出力であるプログラム単位で、リンケージ・エディターに入力する。

オンライン・アプリケーション・プログラム **(online application program).** 表示装置で使用される対話式プログラム。アクティブな場合は、データを待つ。入力が行われると、その入力を処理し、表示装置または別の装置に応答を送信する。

オペレーター・コマンド **(operator command).** 制御プログラムに対するステートメントで、コンソールまたは端末装置から出される。制御プログラムはこれを受けて、要求された情報の提供、通常の操作の変更、新しい操作の開始、または既存の操作の終了を行う。

オプション・ライセンス・プログラム **(optional licensed program).** インストール援助サポートを使用してユーザーが VSE にインストールできる IBM ライセンス・プログラム。

出力パラメーター・テキスト・ブロック **(OPTB) (output parameter text block (OPTB)).** VSE/POWER のスプール・アクセス・サポートにおいて、自動開始のために定義されたユーザー定義のキーワードが * $$ LST ステートメントまたは * $$ PUN ステートメントに含まれている場合に、出力キュー・レコードに含められる情報。

P

ページ・データ・セット **(page data set (PDS)).** ページをプロセッサー・ストレージで必要としない場合に、そのページが記憶されるディスク・ストレージの 1 つ以上のエクステント。

ページの固定 **(page fixing).** ページをマーク付けして、明示的に解除しない限りプロセッサー・ストレージにとどまるようにすること。明示的に解除されるまでは、ページアウトできない。

ページ入出力 **(page I/O).** ページイン操作およびページアウト操作。

ページ・プール **(page pool).** 仮想モードのプログラムのページングに使用可能な、ページ・フレームのセット。

パネル **(panel).** 端末画面上に一度に表示される一式の情報。パネルを上下にスクロールする操作は、マニュアルのページをめくる操作に相当する。「選択パネル *(selection panel)*」も参照。

区画平衡 **(partition balancing).** システムの複数またはすべての区画でプロセッサー上での時間と同じ時間がかかることをユーザーが指定できる z/VSE 機能。

PASN-AL (1 次アドレス・スペース番号 **-** アクセス・リスト**) (PASN-AL (primary address space number - access list)).** 区画と関連付けられているアクセス・リスト。プログラムは、その区画に関連付けられた PASN-AL、およびそのタスク (作業単位) に関連付けられた DU-AL を使用する。「*DU-AL*」 も参照。

各区画には、独自の固有 PASN-AL がある。この区画で実行されるすべてのプログラムが、PASN-AL からデータ・スペースにアクセスできる。したがって、プログラムはデータ・スペースを作成し、それに対する項目を PASN-AL 内に追加し、その項目に索引付けする ALET を取得できる。その区画に含まれている他のプログラムに ALET を渡すことによって、プログラムは、その同じ区画で実行される他のプログラムとデータ・スペースを共有できる。

PDS. ページ・データ・セット。

フェーズ **(phase).** 仮想記憶にロードできる実行可能コードの最小単位。

物理レコード **(physical record).** 補助ストレージとの間で受け渡しされるデータの量。「ブロック *(block)*」と同義。

PNET. VSE/POWER のもとで使用可能なプログラミング・サポート。選択されたジョブ、オペレーター・コマンド、メッセージ、およびプログラム出力をネットワークのノード間で送信できるようにする。

POWER. *VSE/POWER* を参照してください。

事前生成オペレーティング・システム **(pregenerated operating system).** 主としてオブジェクト・コードの形式で IBM から出荷される、z/VSE などのオペレーティング・システム。主制御プログラムのサイズ、ライブラリーの編成とサイズ、およびディスク上に必要なシステム域などの重要な特性は、IBM で定義される。ユーザーはオペレーティング・システムを生成する必要がない。

予防保守 **(preventive service).** 予想される問題を避けるために、1 つ以上の PTF を VSE システムにインストールすること。

1 次アドレス・スペース **(primary address space).** z/VSE において、区画が実行されるアドレス・スペース。1 次モードのプログラムは、1 次アドレス・スペースからデータを取り出す。

1 次ライブラリー **(primary library).** 特定の端末ユーザーが所有し、そのユーザーが直接アクセスできる VSE ライブラリー。

プリンター・キーボード・モード **(printer/keyboard mode).** 1050 コンソール・モードまたは 3215 コンソール・モード (装置依存) を指す。

VSE 印刷サービス機能**/VSE (Print Services Facility (PSF)/VSE).** 高機能プリンターをサポートするアクセス方式。

専用区域 **(private area).** 共有域 (24 ビット) と共有域 (31 ビット) の間の仮想スペース。ここで (専用) 区画が割り振られる。専用区域の最大サイズは、IPL 時に定義される。「共有域 *(shared area)*」 も参照。

専用メモリー・オブジェクト **(private memory object).** メモリー・オブジェクトを作成した区画のみがアクセスできる、2 GB の制限を超えて割り振られるメモリー・オブジェクト (仮想記憶のチャンク)。

専用区画 **(private partition).** 共用として定義されていないシステムのすべての区画。「共用区画 *(shared partition)*」も参照。

実動ライブラリー **(production library).**
1. 事前生成オペレーティング・システム (または製品) において、そのシステム (または製品) のオブジェクト・コードが入っているプログラム・ライブラリー。
2. 通常の処理に必要なデータが入っているライブラリー。「テスト・ライブラリー *(test library)*」 と対比。

プログラマー論理装置 **(programmer logical unit).** 主にユーザー作成プログラム用に使用可能な論理装置。「論理装置名 *(logical unit name)*」を参照。

プログラム一時修正 **(program temporary fix (PTF)).** APAR に記述された 1 つ以上の問題を解決またはバイパスすること。PTF は、現行リリースのプログラムに対する予防保守を目的に IBM ユーザーに配布される。

PSF/VSE. 印刷サービス機能 /VSE (Print Services Facility/VSE)。

PTF. 「プログラム一時修正 *(Program temporary fix)*」を参照。

Q

キュー制御域 **(QCA) (Queue Control Area (QCA)).** VSE/POWER において、データ・ファイルの領域。以下のものが含まれていることがある。
- 拡張チェックポイント情報。
- 共有環境に関する制御情報。

待ち行列ファイル **(queue file).** VSE/POWER によって維持される直接アクセス・ファイル。ジョブ入力およびジョブ出力のスプーリングに関する制御情報が保持されている。

R

ランダム処理 **(random processing).** データの処理を、ディスク・ストレージ上でのデータの位置とは無関係に、データの処理対象となる入力によって決まる任意の順序で行うこと。

実アドレス域 **(real address area).** z/VSE において、動的アドレス変換 (DAT) がオフの状態で、アクセスされる主記憶域。

実アドレス・スペース **(real address space).** アドレスが主記憶域内のアドレスに 1 対 1 でマップされるアドレス・スペース。

実モード **(real mode).** VSE において、プログラムがページングされない処理モード。「仮想モード *(virtual mode)*」 と対比。

リカバリー管理サポート **(RMS) (recovery management support (RMS)).** ハードウェア障害に関する情報を収集し、プロセッサー、入出力装置、またはチャネルのエラーによって失敗した操作の再試行を開始するシステム・ルーチン。

リフレッシュ・リリース **(refresh release).** アップグレードされた VSE システムで、リリースの最新の保守レベルが適用されたもの。

相対レコード・ファイル **(relative-record file).** レコードが固定長スロットにロードされ、各スロットの相対レコード番号に基づいてアクセスされる VSE/VSAM ファイル。

リリース・アップグレード **(release upgrade).** FSU 機能を使用して、z/VSE の新規リリースをインストールすること。

再配置可能モジュール **(relocatable module).** タイプ・オブジェクトのライブラリー・メンバー。1 つのメンバーとしてカタログが作成される 1 つ以上の制御セクションで構成される。

再配置ローダー **(relocating loader).** 必要に応じてフェーズのアドレスを変更し、ユーザーが選択した区画にそのフェーズを実行のためにロードする機能。

リモート・インターフェース **(remote interface).** z/VSE のコンテキストでは、リモート・インターフェースを使用すると、EJB がリモート z/VSE ホスト上にある場合でも、クライアントは EJB に対してメソッド呼び出しを行うことができる。コンテナーはリモート・インターフェースを使用して、クライアント・サイド・スタブおよびサーバー・サイド・プロキシー・オブジェクトを作成し、クライアントから EJB への入力メソッド呼び出しを処理する。

リモート・プロシージャー・コール **(RPC) (remote procedure call (RPC)).**
1. クライアントがサーバーからのプロシージャー呼び出しの実行を要求するために使用する機能。この機能には、プロシージャーのライブラリーおよび外部データ表現が組み込まれている。
2. 別のノード内のサービス・プロバイダーに対するクライアント要求。

常駐モード **(RMODE) (residency mode (RMODE)).** プログラムが仮想ストレージに常駐すると思われる位置を指すプログラム属性。RMODE 24 は、プログラムが 24 ビット・アドレス可能域 (16 メガバイトより下) に常駐しなければならないことを示す。RMODE ANY は、プログラムが 31 ビット・アドレス可能ストレージ (16 メガバイトより上または下) の任意の場所に常駐できることを示す。

REXX/VSE. 汎用プログラミング言語で、特にコマンド・プロシージャー、バッチ・プログラムの高速作成、プロトタイピング、およびパーソナル・ユーティリティーに適している。

RMS. リカバリー管理サポート。

RPG II. ビジネス・データ処理用のアプリケーション・プログラムを作成するために設計された、商用指向のプログラミング言語。

S

SAM ESDS ファイル **(SAM ESDS file).** VSE/VSAM スペースで管理される SAM ファイルであり、したがって SAM マクロと VSE/VSAM マクロの両方によりアクセスできるファイル。

SCP. システム制御プログラミング。

SDL. システム・ディレクトリー・リスト。

検索チェーン **(search chain).** 指定したタイプの特定のライブラリー・メンバーを取得するために、チェーン・サブライブラリーを検索する順序。

第 **2** レベル・ディレクトリー **(second-level directory).** システム・サブライブラリーのディレクトリー・トラックにある最高位のフェーズ名が入っている SVA 内のテーブル。

Secure Sockets Layer (SSL). クライアントがサーバーの認証を受け、すべてのデータおよび要求を暗号化できるようにするセキュリティー・プロトコル。SSL は、Netscape Communications Corp. および RSA Data Security, Inc. によって開発された。

セグメント化 **(segmentation).** VSE/POWER において、プログラムのリスト出力または穿孔出力を複数のセグメントに分割し、このプログラムがそのような出力の生成を終了する前に、印刷または穿孔を開始できるようにする機能。

選択パネル **(selection panel).** 項目のリストが表示された画面で、ユーザーはここから項目を選択できる。「メニュー *(menu)*」と同じ。

検知 **(sense).** 特定の入出力装置または通信装置の状況や特性を (要求に対して、または自動的に) 判別すること。

順次アクセス方式 **(sequential access method (SAM)).** 入出力装置との間で、1 レコードずつまたは 1 ブロックずつ次々に読み書きを行うデータ・アクセス方式。このサポートは、要求に応じて、印刷装置での行送りまたはページ替え、あるいはテープ・ドライブ上でのテープ・マークのスキップなどの装置制御操作を実行する。

サービス・ノード **(service node).** VSE 不在ノード・サポート内で、不在ノードに配布するためにコピーされたマスター VSE システムをインストールしてテストするために使用されるプロセッサー。また、プログラム修正は、まずサービス・ノードで適用されてから、不在ノードに送信される。

サービス・プログラム **(service program).** システムをサポートするために機能を実行するコンピューター・プログラム。「ユーティリティー・プログラム *(utility program)*」 を参照。

サービス・リフレッシュ **(service refresh).** サービスの一形式で、すべてのソフトウェアの現行バージョンが収められている。システム・リフレッシュ とも呼ばれる。

サービス装置 **(service unit).** ディスクまたはテープ (カートリッジ) 上の 1 つ以上の PTF。

共用域 **(shared area).** z/VSE において、共有域 (24 ビット) には監視プログラム域および SVA (24 ビット) が含まれており、共有域 (31 ビット) には SVA (31 ビット) が含まれている。共有域 (24 ビット) はアドレス・スペースの先頭 (16 MB より下) にあり、共有域 (31 ビット) は末尾 (2 GB より下) にある。

共用ディスク・オプション **(shared disk option).** 独立した複数のコンピューター・システムで、共用ディスク装置上の共通データを使用できるようにするオプション。

共有メモリー・オブジェクト **(shared memory objects).** 2 GB の制限を超えて割り振られる仮想記憶のチャンク。複数の区画で共有できる。

共有区画 **(shared partition).** VSE において、システムの仮想アドレス・スペースの他の区画にあるプログラムにサービスを提供し、このようなプログラムと通信を行うプログラム (例えば、VSE/POWER) に割り振られた区画。多くの場合、共有区画は不要。

共用スプーリング **(shared spooling).** VSE/POWER アカウント・ファイル、データ・ファイル、および待ち行列ファイルを、VSE/POWER を用いている複数のコンピューター・システム相互間で共用させる機能。

共用仮想記憶域 **(SVA) (shared virtual area (SVA)).** z/VSE において、頻繁に使用されるフェーズ、区画間で共有されている常駐プログラム、およびシステム・サポート用の区域のリスト・システム・ディレクトリー・リスト (SDL) が入っている高位アドレス域。

SIT (システム初期設定テーブル**) (SIT (System Initialization Table)).** システム初期設定プロセスで使用されるデータが入っている CICS のテーブル。具体的には、SIT は、ロードされる指定済みの CICS システム制御プログラムおよび CICS テーブルのバージョンを (接尾部文字によって) 識別できる。

スケルトン **(skeleton).** ユーザー固有の情報を挿入してはじめて処理できるようになる一連の制御ステートメントまたは命令 (あるいはこれらの両方)。

Socks 化された **(socksified).** 「*Socks* 対応 *(socks-enabled)*」 を参照。

Socks 対応 **(Socks-enabled).** *Socks* プロトコル を認識する TCP/IP ソフトウェアまたは特定の TCP/IP アプリケーションに関係する用語。「Socks 化された (Socksified)」は、「Socks 対応」を表す俗語。

Socks プロトコル **(socks protocol).** 保護されたネットワーク内のアプリケーションが、*Socks* サーバー 経由でファイアウォールを通過して通信できるようにするプロトコル。

Socks サーバー **(socks server).** 保護されていないネットワーク内のサーバー・アプリケーションに、ファイアウォール経由のセキュアな片方向接続を提供する回線レベル・ゲートウェイ。

ソース・メンバー **(source member).** VSE がサポートするいずれかのプログラミング言語で書かれたソース・ステートメントを含むライブラリー・メンバー。

分割 **(split).** 空きスペースの指定最小数が、新しいレコードで使い果たされたときに、動的にストレージ・スペース (CI または CA) の指定された単位を 2 倍にすること。

スプーリング **(spooling).** ディスク装置をバッファー・ストレージとして使用し、コンピューターの周辺装置とプロセッサーの間のデータ転送時の処理遅延を少なくすること。z/VSE では、スプーリングは VSE/POWER の制御下で行われる。

スプール・アクセス保護 **(Spool Access Protection).** VSE/POWER のオプション機能。個々のスプール・ファイル項目アクセスを、セキュリティー・ログオンの実行によって認証されたユーザー ID に制限する。

スプール・ファイル **(spool file).**
1. 後で処理するために保存された出力データが入っているファイル。
2. ディスク上の 3 つの VSE/POWER ファイル (キュー・ファイル、データ・ファイル、およびアカウント・ファイル) のうちの 1 つ。

SSL. 「Secure Sockets Layer」を参照。

スタック・テープ **(stacked tape).** IBM 提供の製品付属のテープで、複数のライセンス・プログラムのコードが入っている。

標準ラベル **(standard label).** テープ・リールなどのデータのボリューム、またはデータのボリュームの一部であるファイルを識別する固定形式のレコード。

独立型プログラム **(stand-alone program).** VSE システムから独立して (制御されずに) 稼働するプログラム。

スタートアップ **(startup).** オペレーティング・システムの IPL を実行して、すべてのサブシステムとアプリケーション・プログラムを操作可能にする処理。

開始オプション **(start option).** VTAM において、VTAM システムの作動時の条件を決定するユーザー指定のオプションまたは IBM 指定のオプション。開始オプションは事前定義することも、VTAM の開始時に指定することもできる。

静的区画 **(static partition).** IPL 時に定義される区画で、定義された一定量の仮想記憶域を占有する。「動的区画 *(dynamic partition)*」 も参照。

ストレージ・ディレクター **(storage director).** 記憶制御装置の独立したコンポーネント。記憶制御装置のすべての機能を実行するため、この装置に接続されたディスク装置へのアクセス・パスを 1 つ提供する。1 台の記憶制御装置には 2 つのストレージ・ディレクターが備えられている。

ストレージのフラグメント化 **(storage fragmentation).** 仮想記憶域の実アドレス範囲または仮想アドレス範囲で、ストレージの未使用セクション (フラグメント) を割り振ることができないこと。

副次割り振りされたファイル **(suballocated file).** すでに定義済みのデータ・スペースの一部を占める VSE/VSAM ファイル。このデータ・スペースには、他のファイルを入れることができる。「固有ファイル *(unique file)*」 も参照。

サブライブラリー **(sublibrary).** VSE において、ライブラリーをさらに分割した一部分。サブライブラリー内でのみ、メンバーにアクセスできる。

サブライブラリー登録簿 **(sublibrary directory).** アクセスしたサブライブラリー内でメンバーを見つけるためのシステム用索引。

サブミット **(submit).** 処理のためにジョブをシステムに渡す VSE/POWER 機能。

SVA. 「共用仮想記憶域 (shared virtual area)」を参照。

同期データ・リンク制御 **(SDLC) (Synchronous DataLink Control (SDLC)).** リンク接続による同期、コード透過、ビット・シリアル情報転送を管理するための規則。伝送交換は、交換リンクまたは非交換リンク上で全二重または半二重で行われる。リンク接続の構成は、Point-to-Point、マルチポイント、またはループのいずれかになる。

SYSRES. 「システム常駐ボリューム (system residence volume)」を参照。

システム制御プログラミング **(SCP) (system control programming (SCP)).** システムの操作またはそのサービス、あるいはその両方の基礎となる、IBM 提供の非ライセンス・プログラム。

システム・ディレクトリー・リスト **(SDL) (system directory list (SDL)).** 使用頻度の高いフェーズおよび SVA に常駐するすべてのフェーズのディレクトリー項目を含むリスト。このリストは SVA に入っている。

システム・ファイル **(system file).** z/VSE において、オペレーティング・システムが使用するファイル。例えば、ハードコピー・ファイル、記録ファイル、ページ・データ・セットなど。

システム初期設定テーブル **(SIT) (System Initialization Table (SIT)).** システム初期設定プロセスで使用されるデータが入っている CICS のテーブル。具体的には、SIT は、ロードされる指定済みの CICS システム制御プログラムおよび CICS テーブルのバージョンを (接尾部文字によって) 識別できる。

システム記録ファイル **(system recorder file).** ハードウェアの信頼性データを記録するために使用されるファイル。「記録ファイル *(recorder file)*」 と同じ。

システム・リフレッシュ **(system refresh).** 「サービス・リフレッシュ *(service refresh)*」 を参照。

システム・リフレッシュ・リリース **(system refresh release).** 「リフレッシュ・リリース *(refresh release)*」を参照。

システム常駐ファイル **(SYSRES) (system residence file (SYSRES)).** オペレーティング・システムを収めた z/VSE システム・サブライブラリー IJSYSRS.SYSLIB。このファイルは、システム常駐ボリューム DORSES に格納されている。

システム常駐ボリューム **(SYSRES) (system residence volume (SYSRES)).** システム・サブライブラリーが保管されているディスク・ボリューム。ここから、ハードウェアがシステム始動用の初期プログラム・ロード・ルーチンを取得する。

システム・サブライブラリー **(system sublibrary).** オペレーティング・システムが入っているサブライブラリー。システム常駐ボリューム (SYSRES) に保管される。

T

タスク管理 **(task management).** タスクによるプロセッサーおよび他のリソース (入出力装置を除く) の使用を制御する制御プログラムの機能。

時間イベント・スケジューリング・サポート **(time event scheduling support).** VSE/POWER では、時間イベント・スケジューリング・サポートにより、事前定義の時刻に 1 回または繰り返してジョブを区画内で処理するようにスケジュールできる。* $$ JOB ステートメントの時間イベント・スケジューリング・オペランドを使用して、必要なスケジューリング時刻を指定する。

TLS. 「Transport Layer Security」を参照。

トラック・グループ **(track group).** VSE/POWER において、CKD 装置用のファイルの基本組織単位。

トラック保護 **(track hold).** あるプログラムによって更新されているトラックが、別のプログラムによってアクセスされないように保護する機能。

トランザクション **(transaction).**
1. バッチまたはリモート・バッチ入力における、ジョブまたはジョブ・ステップ。2. CICS TS では、表示装置のオペレーターが使用できる 1 つ以上のアプリケーション・プログラム。ある 1 つのトランザクションは、1 つまたは複数のディスプレイ装置から同時に使用することができる。特定のオペレーターに対するトランザクションの実行は、タスクとしても参照される。
2. 与えられたタスクは、一人のオペレーターのみに関連づけることができる。

一時域 **(transient area).** 要求に応じて優先順位の高いシステム・サービスを提供するために使用される制御プログラム内部の区域。

Transport Layer Security. 最新の SSL 暗号プロトコル。プライバシーおよびデータ保全性にさらに強度を提供します。

ターボ・ディスパッチャー **(Turbo Dispatcher).** マルチプロセッサー・システム (CEC (中央電子複合システム) ともいう) を使用できる z/VSE の機能。そのような CEC 内の各 CPU は、z/VSE の共有仮想記憶域 (監視プログラム、共有域 (24 ビット)、および共有域 (31 ビット)) にアクセスできる。すべての CPU には、同等の権限がある。つまり、CPU は割り込みを受信し、特定の CPU が複数の作業単位を占有することはない。

U

UCB. 汎用文字セット・バッファー (Universal character set buffer)。

汎用文字セット・バッファー **(UCB) (universal character set buffer (UCB)).** UCS 情報を保持するためのバッファー。

UCS. 汎用文字セット (Universal character set)。

ユーザー・コンソール **(user console).** z/VSE において、特定のコンソールに対するシステム・メッセージのみ受信するコンソール。それらのメッセージは、例えば、メッセージをコンソールにエコーさせるための要求とともにサブミットされるジョブから出される。「マスター・コンソール *(master console)*」 と対比。

ユーザー出口 **(user exit).** IBM のソフトウェア・プロダクトが提供するプログラミング・サービスであり、アプリケーション・プログラムの実行中に要求して、後でユーザー指定のイベントが発生したときに制御をアプリケーション・プログラムに戻すことができる。

V

可変長相対レコード・データ・セット **(VRDS) (variable-length relative-record data set (VRDS)).** 可変長レコードが含まれている の相対レコード・データ・セット。「相対レコード・データ・セット *(relative-record data set)*」も参照。

可変長相対レコード・ファイル **(variable-length relative-record file).** 可変長レコードが含まれている VSE/VSAM の相対レコード・ファイル。「相対レコード・ファイル *(relative-record file)*」も参照。

VIO. 「仮想入出力域 (virtual I/O area)」を参照。

仮想アドレス **(virtual address).** 仮想記憶内の場所を指し示すアドレス。仮想アドレスに記憶されている情報を使用するときには、システムによってプロセッサー・ストレージ・アドレスに変換される。

仮想アドレス可能度拡張 **(VAE) (virtual addressability extension (VAE)).** 複数の仮想アドレス・スペースの使用を許可するストレージ管理サポート

仮想アドレス・スペース **(virtual address space).** 仮想アドレス域 (仮想ストレージ) のサブディビジョンであり、ユーザーが専用区画および非共有区画を割り振ることができる。

仮想ディスク **(virtual disk).** プログラムが作業スペースとして使用できる最大 2GB の連続仮想ストレージ・アドレスの範囲。仮想ディスクはストレージに存在するが、ユーザー・プログラムからは実 FBA ディスク装置と同様に扱える。仮想ディスクに対するすべての入出力操作はインターセプトされ、ディスクに対する書き込みまたは読み取りデータはデータ・スペースとの間で移動される。

データ・スペースと同様、仮想ディスクにはユーザー・データだけを書き込むことができる。共用域、システム・データ、またはシステム・プログラムを書き込むことはできない。アドレス・スペースまたはデータ・スペースと異なり、データは仮想ディスク上では直接アドレス指定できない。仮想ディスクのデータを操作する場合、プログラムは入出力操作を実行しなければならない。

z/VSE 5.2 からは、仮想ディスクは共有メモリー・オブジェクトで定義できる。

仮想入出力域 **(VIO) (virtual I/O area (VIO)).** ページ・データ・セットの拡張。主に制御データ用の中間ストレージとしてシステムによって使用される。

仮想モード **(virtual mode).** プログラムの操作モード。プログラムの仮想記憶域を補助するために必要なプロセッサーの (実) 記憶域が不足した場合に、仮想記憶域をページングできる。

仮想区画 **(virtual partition).** VSE において、仮想記憶の動的区域の一部。

仮想ストレージ **(virtual storage).** ユーザーのためのアドレス可能スペース・イメージで、そこから命令とデータがプロセッサー・ストレージ内の該当位置にマッピングされる。

仮想テープ **(virtual tape).** z/VSE において、仮想テープは、テープ・イメージを含むファイル (またはデータ・セット) である。仮想テープでは、物理テープと同じ方法で読み取りまたは書き込みを行うことができる。仮想テープには以下のようなものがある。

- z/VSE ローカル・システムの VSE/VSAM ESDS ファイル。
- サーバー・サイドのリモート・ファイル。例えば、Linux、UNIX、または Windows のファイル。そのようなリモート仮想テープにアクセスするには、z/VSE とリモート・システムとの間の TCP/IP 接続が必要。

ボリューム **ID (volume ID).** ボリューム通し番号。ボリュームをシステムで使用できるように準備するときにボリューム内に割り当てられる番号。

VRDS. 可変長相対レコード・データ・セット。「可変長相対レコード・ファイル *(variable-length relative record file)*」 を参照。

VSAM. *VSE/VSAM* を参照してください。

VSE (拡張仮想記憶) (Virtual Storage Extended). 基本オペレーティング・システムおよびユーザーが必要とするデータ処理のための IBM 提供プログラムおよびユーザー作成プログラムから構成されるシステム。VSE および VSE が制御するハードウェアは、1 個の完結したコンピューティング・システムを形成することになる。現行バージョンは z/VSE と呼ばれる。

VSE 拡張機能 **(VSE/Advanced Functions).** 基本システム制御を提供するプログラムで、ライブラリアン、リンケージ・エディターなどの監視プログラムおよびシステム・プログラムを含む。

VSE コネクター・サーバー **(VSE Connector Server).** VSE JavaBeans のホスト部分であり、z/VSE のインストール時に読み取りキューに入れられるジョブ STARTVCS を使用して開始される。デフォルトで動的クラス R で実行される。

VSE/DITTO (VSE/ファイル間データ転送、テスト、および操作ユーティリティー**) (VSE/DITTO (VSE/Data Interfile Transfer, Testing, and Operations Utility)).** ディスク装置、テープ装置、およびカード装置用のファイル間サービスを提供する IBM ライセンス・プログラム。

VSE/ESA (Virtual Storage Extended/Enterprise Systems Architecture). z/VSE の以前に使用されていたシステム。

VSE/ 高速コピー **(VSE/Fast Copy).** ディスクからディスクへの高速コピー・データ操作用、および磁気テープまたはディスク上の中間ダンプ・ファイルを介したダンプ/復元操作用のユーティリティー・プログラム。

VSE/FCOPY (VSE/データ・セット高速コピー・プログラム**) (VSE/FCOPY (VSE/Fast Copy Data Set program)).** ディスクからディスクへの高速コピー・データ操作用、または磁気テープまたはディスク上の中間ダンプ・ファイルを介したダンプ/復元操作用の IBM ライセンス・プログラム。独立型バージョンの FASTCOPY ユーティリティーもある。

VSE/ICCF (VSE/対話式計算制御機能**) (VSE/ICCF (VSE/Interactive Computing and Control Facility)).**

システムのプロセッサーにリンクされた端末の許可ユーザーに対して、タイム・スライス・ベースでインターフェースの役割を果たす IBM ライセンス・プログラム。

VSE/ICCF ライブラリー (VSE/ICCF library). システム・データおよびユーザー・データが入っている小さなファイル (ライブラリー) から構成されるファイルで、VSE/ICCF の制御下でアクセスできる。

VSE JavaBeans. すべての VSE ベースのファイル・システム (VSE/VSAM、ライブラリアン、および VSE/ICCF) へのアクセスを許可して、ジョブを実行依頼し、z/VSE オペレーター・コンソールにアクセスする JavaBeans。クラス・ライブラリーは *VSEConnector.jar* アーカイブ内に含まれている。「*JavaBeans*」も参照。

VSE ライブラリー (VSE library). ディスク上に保管された、各種形式のプログラムとストレージ・ダンプの集合。プログラムの形式は、ソース・コード、オブジェクト・モジュール、フェーズ、またはプロシージャーなどのメンバー・タイプによって示される。VSE ライブラリーは、あらゆるタイプのメンバーが入っている最低 1 つのサブライブラリーで構成される。

VSE/POWER. 主として入出力をスプールするために使用される IBM ライセンス・プログラム。このプログラムのネットワーキング機能により、VSE システムは、他のリモート・サーバーとファイルを交換したり、あるいは他のリモート・プロセッサーでジョブを実行することができる。

VSE/VSAM (VSE/仮想記憶アクセス方式) (VSE/Virtual Storage Access Method). 磁気ディスク装置上にある固定長レコードと可変長レコードの直接処理または順次処理のための IBM アクセス方式。

VSE/VSAM カタログ (VSE/VSAM catalog). ファイルとボリュームに関する包括的な情報が入っているファイル。ファイルの探索、ストレージ・スペースの割り振りと割り振り解除、プログラムまたはオペレーターがファイルにアクセスする許可をもっているかどうかの検査、およびファイルの使用統計の累積の際、VSE/VSAM はこのカタログを必要とする。

VSE/VSAM 管理スペース (VSE/VSAM managed space) . VSE/VSAM の管理下に置かれている、ディスク上のユーザー定義のスペース。

W

実行待ちサブキュー **(wait for run subqueue).** VSE/POWER において、ディスパッチ可能ジョブが実行開始時間の順序で並べられた、読み取りキューのサブキュー。

待ち状態 **(wait state).** すべての操作が延期されたときの、プロセッサーの状態。ハードウェア・ウェイト状態からシステムをリカバリーするには、新規システムのスタートアップを実行させる必要がある。「ハード待ち *(hard wait)*」 を参照。

ワークステーション・ファイル転送サポート **(Workstation File Transfer Support).** データが中間ストレージに保管されている z/VSE ホスト・システムとリンクで結ばれた IBM パーソナル・コンピューター (PC) 間のデータ交換を可能にするサポート。PC のユーザーはこのデータを取得でき、z/VSE とは無関係に処理できる。

作業ファイル **(work file).** 処理中のデータの一時保管のために使用されるファイル。

数字

24 ビット・アドレッシング **(24-bit addressing).** 最大 16MB のアドレス・スペースのアドレス可能度を提供するもの。

31 ビット・アドレッシング **(31-bit addressing).** 最大 2GB のアドレス・スペースのアドレス可能度を提供するもの。

64 ビット・アドレッシング **(64-bit addressing).** 最大 2 ギガバイト以上のアドレス・スペースのアドレス可能度を提供する。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[数字]

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

プログラム番号: 5686-VS6

SC43-2945-01

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21