組込みシステム開発専用 TCP/IP プロトコルスタック

KASAGO TCP/IP

ユーザーズマニュアル

簡易版

図研エルミック株式会社

改版履歴

版	日付	内容
1.0	2014/06/04	新規作成

目次

1. はじめに

本マニュアルはKASAGO TCP/IPで使用できる機能をご紹介するために、実際の通信アプリケーションで使用するAPIや、簡単な通信のコード例について記載しております。
なお、本マニュアルは無償配布版のため、APIについては通信アプリケーションを作成する上で一般的に使用される関数のみ記載しております。

2. BSD ソケットについて

BSD ソケットとは

バークレーソケット4.4API(アプリケーションプログラマーインターフェース) は、標準ファンクションコールの一式であり、アプリケーションレベルで使用できます。プログラマーは、これらの関数を使用して、製品にインターネット通信機能を組込むことができます。

バークレーソケットAPI(以後、ソケットとも言う)は、1983年に4.2BSDとしてリリースされ、4.4BSDまで拡張されています。バークレーベースコードは、コマーシャル、パブリック共に、BSD/OS, FreeBSD, NetBSD,OpenBSD, およびUnixWare2.x等の各種オペレーティングシステムで使用されています。その他、SolarisおよびLinux のようなオペレーティングシステムでは、最初からコードが書かれていますが、標準ソケットインターフェースを採用しています。

通常はバークレーソケットが標準とされますが、他のソケットAPIもあります。最も知られているAPIは、WinsockとTLIです。Winsock(Windowsソケット)は、1993年にMicrosoft Windowsプラットフォーム向けに開発され、BSD インターフェースに忠実に準拠したものです。ほとんどの例外がBSDシステムのプラットフォームに特有であって、BSD APIの大きいサブセットが提供されます。TLI (TransportLayer Interface)は、AT&T社によって開発され、TCP/IPおよびIPX/SPXトランスポートレイヤーにアクセスする機能があります。

XTI (X/Open Transport Interface, developed byX/Open CompanyLtd開発)は、TLIの拡張版でTCP/IPおよびNetBiosへのアクセスが可能です。

ソケット関数の概要

BSDソケットは通常クライアント・サーバーアーキテクチャに依存します。TCP通信には、単一ホストが受信接続リクエストをキャッチします。リクエストが届くと、サーバーホストが受信し、その時点でデータはホスト間を移動できるようになります。UDPの場合は、これがなくても接続を確立することができ、ホストからのデータ送受信が可能となります。

ソケットAPIは、データをアプリケーションレベルのポートとソケットに送信するために、2種類のメカニズムを利用します。ポートとソケットのコンセプトは、ソケットプログラミングにおいて、誤解されがちです。

すべてのTCP/IPスタックは、TCPおよびUDPに対し、65,536のポートを持っています。UDP(0-65535)対応のポートがあり、また、同じナンバリングスキームでTCP対応のポートがあります。これらは、重複することはありません。このようにして、TCPおよびUDPの通信が、例えば、ポート15上で同時に実行されます。

ポートは物理インターフェースではなく、いわゆるヒューマンインターネットコミュニケーションコンセプトを簡略化したものです。パケットを受信すると、プロトコルスタックが指定ポートに導きます。ポート上

で接続待ち(listen)しているアプリケーションがない場合、パケットは処分され、エラーは送信者に返信されますが、アプリケーションはソケットがポートに付加できるようなソケットを作成できます。アプリケーションがソケットを作成し、ソケットをポートに送り込むと、そのデータがアプリケーションに送信されます。外部(ポート)とアプリケーションを結ぶメカニズムであるソケットという言葉が使用される訳はここにあります。

ソケットベースのシステムは他のソケットベースのシステムとしか通信できないと誤解されますが、TCP/IPまたはUDP/IP通信はポートレベルで処理されます。基本的なプロトコルはポート上のメカニズムの存在は考慮しません。バークレーソケット、WinSock等、どんなインターネットホストでも通信は可能です。ソケットはプログラマーがインターネット機能にアクセスできるAPIであり、通信方法を修正できません。

ソケットとポートの関係を、オフィスビルの例をあげて説明しましょう。ビルそのものは、インターネットホストに例えられます。各オフィスはポート、受付担当者はソケット、取引ビジネスがアプリケーションです。このビルを訪問するとします。ビルに入り、目的のオフィスに向かいます。オフィスに入り、取引ビジネスへの仲介となる受付担当者に問い合わせます。オフィスに誰もいなければ、オフィスを出ることになります。上記をソケットに置換えてみましょう。パケットはホストに転送され、最終的に正しいポートに辿り着きます。そこで、ソケットはパケットデータをアプリケーションに伝えます。ソケットがなければ、パケットは処分されます。

バイト-オーダリング関数

TCP/IPは統一標準であるため、どんなプラットフォーム間でも通信は可能です。ビッグエンディアンとリトルエンディアンが相互に理解できるよう、情報の調整方法を確立する必要があります。このように、ここでは、ネットワークバイトオーダーでデータのやり取りを実現できる機能があります。

データがすでに適切に配列されているプラットフォームでは、機能する必要がなく、そしてマクロに空のステートメントが入ります。バイトオーダー関数はすでに適切に配列されている場合システム性能に影響を与えず、コードの移植性を向上させるため、常に使用されるべきものです。

4バイトオーダー関数は、htons, htonl, ntohs, およびntohlです。それぞれ、network shortに対するホスト、network longに対するホスト、host shortに対するネットワーク、host longに対するネットワークをそれぞれ意味しています。

htons は、ホストバイトからネットワークバイトオーダーへshort整数を変換します。htonl は、同様に、long整数を変換します。その他二つの関数は、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。

データ構造

実際のAPI 関数を説明するまえに、いくつかの構造体を理解しておく必要があります。
中でもsockaddr_inは最も大切な構造体であり、以下のように定義されます。

```
struct                  sockaddr_in
{
      u_char            sin_len;
      u_char            sin_family;
      u_short           sin_port;
      struct in_addr    sin_addr;
      char              sin_zero[8];
};
```

sockaddr_inに使用される構造体in_addrは、以下のように定義されます。

```
struct       in_addr
{
      u_long s_addr;
};
```

ソケット内で使用されるデータ構造の中で一番重要なものです。後者は、ソケットと関連するＩＰ アドレスを含む符号無しlong整数から構成されます。前者には、sin_familyおよびsin_portという2つの重要なフィールドがあります。
sin_familyは、使用するプロトコルファミリーを示します。IPv4には、常に定数AF_INETが渡されます。
sin_portは、どのポート番号がソケットと関連付けられるかを示します。

sockaddr_in は、標準sockaddr 構造体の修正です。

```
struct      sockaddr
{
    u_char sa_len;
    u_char sa_family ;
    char    sa_data[14] ;
};
```

ソケットコールは、標準sockaddr構造体が望まれますが、IPv4通信の場合には、sockaddrに割当てられたsockaddr_in構造体に渡されるのが適切です。

一般的なソケットコール

ここでは、よく使用されるソケットコールと、その使用法を説明します。この説明は概要レベルにとどまるため、コールの機能に関しては、本マニュアルの「プログラマーズリファレンス」を参照してください。

socket

ソケットとは、簡単に言えば、ソケットAPIによって使用されるデータ構造のことです。この関数を呼び出すと、ソケットを作成し、ソケットの参照番号を返します。その参照番号は、今後のコールに使用されます。

bind

このコールによって、ソケットと特定のローカルポートおよびIPアドレスを関連付けることができます。（以下のlistenおよびacceptを参照してください） ユーザーは、外部からの接続に対応するポートおよびIPアドレスを指定できます。

外部への接続リクエストに対しては、(以下のconnectを参照のこと) 他のホストによって参照されている際に、送信元のポートを指定できます。

注: 外部からの接続を受け取るように設定されていないソケットには、bindは必要ありません。この場合、スタックは適切なI Pアドレスとランダムポート（通称エーテルポート）を選択します。

listen

外部から来るTCP リクエストを受信する指定ソケットを作成します。acceptの前に呼び出されます。

accept

リスニングソケット上の受信接続リクエストを検出します。このコールによって、ブロッキングモードでは、接続リクエストが受信されるまで、スリープ状態となります。ノンブロッキングモードでは、接続リク

エストがないことを示すTM_EWOULDBLOCKを返却し、acceptが再び呼び出されます。ユーザーがacceptを呼び出し、接続要求がペンディング中の場合、acceptはリスニングソケットのプロパティに基づいて別のソケットを作成します。関数が正常終了したら、ソケットディスクリプタが新規に作成され、接続ソケットが返されます。新規ソケットが作成されると、サーバー上の単一ポートと複数のクライアントとの通信が可能となります。(初期設定により、ポート80上で接続待ちし、同時に何千のホストと通信できるウェブサーバーを考えて下さい。ユーザーがacceptを呼び出す度に、接続リクエストのペンディングがあれば、新規ソケットを作成します。)

connect

connectコマンドを発行した場合、スタックは別のホストとの接続を確立します。connectによって、スタックが接続を確立する前に、ソケットおよび送信先IP アドレスおよびポートを保持するsockaddr_in 構造体を渡す必要があります。TCPでは、実際の接続が取り決められますが、パケットは交換されません。

send

接続ソケットでデータを送信できます。sendtoとは異なり、このソケットの接続は必須です。すでに、接続されているため、送信先アドレスを指定する必要はありません(送信先アドレスは、acceptまたは connect に設定されます)。sendはUDPまたはTCPデータに使用できます。

sendto

sendと異なり、sendtoでは、送信先のポートとアドレスを指定する必要があります。TCPは、既存の接続を必要とするので、UDP通信のみ役に立ちます。sendtoは、接続の有無にかかわらず、UDPソケット上で使用できます。UDPソケットが既に接続されている場合、sendtoの送信先アドレスによってconnectでソケット上に設定されたデフォルトが無効になります。

recv

接続ソケットからデータを受信でき、TCP あるいは UDP に使用できます。

recvfrom

指定UDPソケット(接続の有無にかかわらず)からデータを受信できます。TCPソケットは、接続を必

要とするので使用できません。

close

socketコールに割当てられているソケットを閉じ(読み込み:削除) ます。ソケットが接続している場合、削除する前に接続を閉じます。closeコールはその他の目的にも多数使用されるため(例:開いているファイルを閉じる)、KASAGOTCP/IPスタックでは、tfCloseと改名され、既存の関数との混同を防ぎます。

コード例

ソケットAPIを使用して、アプリケーション内にインターネット接続を確立する例を以下にあげます。例はprotocols CDのディレクトリexamples¥で利用できます。例には、UDPクライアント, UDPサーバー,TCPクライアント, およびTCPサーバーの計4つがあります。これらはすべてブロッキングモードでコーディングされています。

UDP クライアント

UDPクライアントのコーディングの方法を示します。ソケットが作成され、sendtoが指定回数呼び出されます。bindは呼び出されないことに留意してください。送信接続では、スタックがランダムポートと適切な
IPアドレスを選択するため、bind は必要ありません。

```
#include <trsocket.h>

#define TM_BUF_SIZE 1400
#define TM_PACKETS_TO_SEND 10
#define TM_DEST_ADDR “10.0.0.1”
#define TM_DEST_PORT 9999

char testBuffer[TM_BUF_SIZE];
char * errorStr;

int   UDPClient(void)
{
      int testSocket;
      unsigned int counter;
      struct sockaddr_in destAddr;
      int errorCode;
      int returnVal;

      counter = 0;
      returnVal = 0;
```

```
/* アドレスファミリーを指定する。*/
    destAddr.sin_family = AF_INET;
/* 送信先のポートを選択する。*/
    destAddr.sin_port = htons(TM_DEST_PORT);
/* 送信先の IP アドレスを選択する*/
    destAddr.sin_addr.s_addr = inet_addr(TM_DEST_ADDR);

/* ソケットを作成する*/
    testSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

/*
 * ソケットが正確に作成されたか確認する。作成されていなければ、ただちに返す。
 */
    if (testSocket == TM_SOCKET_ERROR)
    {
        returnVal = tfGetSocketError(testSocket);
        errorStr = tfStrError(returnVal);

        goto UDPClientEnd;
    }

/* 十分なパケットをまだ送信していない場合*/
    while (counter < TM_PACKETS_TO_SEND)
    {
    /* 上記指定の送信先にもうひとつのパケットを送信する。*/
        errorCode = sendto(testSocket, testBuffer, TM_BUF_SIZE, 0, &destAddr, sizeof(destAddr));
    /*
     * 送信中のエラーがないか確認する。エラーがあれば、ループから抜ける。
     */
        if (errorCode < 0)
        {
            returnVal = tfGetSocketError(testSocket);
            errorStr = tfStrError(returnVal);
            break;
        }
```

```
        /* 送信されたパケットの数をインクリメントする。*/
            counter++;
        }
        UDPClientEnd:
/* 閉じる前に、ソケットが存在するか確認する。*/
        if (testSocket != -1)
        {
        /* ソケットを閉じる。*/
            tfClose(testSocket);
        }
        return(returnVal);
}
```

UDP サーバー

このコードはたいへん簡易なUDPサーバーで、ソケットを作成し、指定のポートにバインドし、データを受信します。(スタックが取得するため、IPアドレスを提供する必要はありません。コードの移植性を向上させるの役に立ちます。)データを受信すると、sourceAddr構造体に、元のIPアドレスおよび受信パケットのポートが書き込まれます。

```
#include <trsocket.h>

#define TM_BUF_SIZE 1500
#define TM_DEST_PORT 9999

char testBuffer[TM_BUF_SIZE];
char * errorStr;

int UDPServer(void)
{
    int                 testSocket;
    struct sockaddr_in sourceAddr;
    struct sockaddr_in destAddr;
    int                 errorCode;
    int                 addrLen;
    int                 returnVal;

    returnVal = 0;
/* アドレスファミリーを指定する。*/
    destAddr.sin_family = AF_INET;
/*
    送信先のポートを選択する。
    (サーバーであるため、送信先ポートはユーザーのバインド先である。)
*/
    destAddr.sin_port = htons(TM_DEST_PORT);

/*
    送信先のIPアドレスを選択する。この値を0に設定すると、どんなIPアドレスを使用するか考慮せ
```

ずに、ひとつ選択することになる。IPアドレスをひとつだけ持つシステムにとっては、もっとも簡単なアプローチである。
*/
```
    destAddr.sin_addr.s_addr = 0;

/*
 * 3番目の値は、使用を希望するプロトコルである。スタックは2番目のパラメータ(SOCK_DGRAM = UDP, SOCK_STREAM = TCP)に基づき、どのプロトコルが使用されるべきかを理解できるため、0 内に渡す。
*/
    testSocket = socket(AF_INET, SOCK_DGRAM, 0);

/* ソケットが適切に作成されたか確認する。*/
    if (testSocket == TM_SOCKET_ERROR)
    {
        returnVal = tfGetSocketError(testSocket);
        errorStr = tfStrError(returnVal);
        goto UDPServerEnd;
    }

/*
 * ソケットを、データを受信するポートおよびアドレスでバインドする。
 *
*/
    errorCode = bind(testSocket, &destAddr, sizeof(destAddr));

/* bindのエラーをチェックする。*/
    if (errorCode < 0)
    {
        returnVal = tfGetSocketError(testSocket);
        errorStr = tfStrError(returnVal);
        goto UDPServerEnd;
    }

/* 継続する*/
    while (1)
```

```
    {
    /* sockaddr_in構造体のサイズを取得する。*/
        addrLen = sizeof(sourceAddr);

    /*
     * データを受け取る。渡される値は、以下の通り。
     * testSocketで該当データを受け取る。
     * データはtestBuffer に格納される。
     * TM_BUF_SIZE バイトまで受信できる。
     * 特に設定するフラグはない。
     * sourceAddr から来たデータをIPアドレス/ポートに格納する。
     * sourceAddrに格納されているデータ長をaddrLenに格納する。
     * addrLenが渡された時、設定された長さを使用して、スタックがsourceAddrに指定以上のバイト
       を書き込まないようにする。
     */

        errorCode = recvfrom(testSocket, testBuffer, TM_BUF_SIZE, 0, &sourceAddr, &addrLen);

    /* recvfrom にエラーがないか確認する。*/
        if (errorCode < 0)
        {
            returnVal = tfGetSocketError(testSocket);
            errorStr = tfStrError(returnVal);
            break;
        }
    }
    udpServerEnd:
/* 閉じる前に、ソケットが存在するか確認する。*/
    if (testSocket != -1)
    {
/* ソケットを閉じる。*/
        tfClose(testSocket);
    }
    return(returnVal);
}
```

TCP クライアント

本コードは、UDPクライアントと類似していますが、UDPと異なり、TCPにはネゴシエーションが必要なため、実際にデータを転送する前にconnectを呼ぶ必要があります。次に、送信を指定された回数呼び出します。実際にワイヤー上で送信されるTCP データパケット数はこのコードで定義される数と大抵異なります。TCP は、データグラムベースというより、ストリームベースなので、データをバッファし、もっとも適切なサイズパケットで（一般的に最大サイズ）送信します。

```
#include <trsocket.h>

#define TM_BUF_SIZE 1400
#define TM_PACKETS_TO_SEND 10
#define TM_DEST_ADDR “10.129.36.52”
#define TM_DEST_PORT 9999

char testBuffer[TM_BUF_SIZE];
char * errorStr;

int tcpClient(void)
{
    int                 testSocket;
    unsigned int        counter;
    struct sockaddr_in destAddr;
    int                 errorCode;
    int                 sockOption;
    int                 returnVal;

    returnVal = 0;
    counter = 0;

/* struct sockaddr_inのデータの長さ */
    destAddr.sin_len = sizeof(sockaddr_in);

/* アドレスファミリーを指定する。*/
    destAddr.sin_family = AF_INET;
```

```
/* 送信先のポートを選択する。*/
    destAddr.sin_port = htons(TM_DEST_PORT);

/* 送信先のIP アドレスを選択する*/
    destAddr.sin_addr.s_addr = inet_addr(TM_DEST_ADDR);

/* ソケットを作成する。*/
    testSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

/*
 * ソケットが正確に作成されたか確認する。作成されなければ、直ちに戻ります。
*/
    if (testSocket == TM_SOCKET_ERROR)
    {
        returnVal = tfGetSocketError(testSocket);
        errorStr = tfStrError(returnVal);
        goto tcpClientEnd;
    }
/* サーバーに接続する*/
    errorCode = connect(testSocket, &destAddr, sizeof(destAddr));

/* 適切に接続されたか確認する。*/
    if (errorCode < 0)
    {
        returnVal = tfGetSocketError(testSocket);
        errorStr = tfStrError(returnVal);
        goto tcpClientEnd;
    }

/* 十分なパケットをまだ送信していない場合 */
    while (counter < TM_PACKETS_TO_SEND)
    {
    /* 上記で指定された場所へ別のパケットを送る。*/
        errorCode = send(testSocket, testBuffer, TM_BUF_SIZE, 0);
    /* 送信中のエラーがないか確認する。エラーがあれば、ループから抜ける。 */
```

```
            if (errorCode < 0)
            {
                returnVal = tfGetSocketError(testSocket);
                errorStr = tfStrError(returnVal);
                break;
            }
        /* 送信されたパケット数をインクリメントする。*/
            counter++;
        }
        tcpClientEnd;
/* 閉じる前に、ソケットが存在するか確認する。 */
        if (testSocket != -1)
        {
            tfClose(testSocket);
        }
        return(returnVal);
}
```

TCP サーバー

これは、最も複雑なサンプルです。ソケットを作成し、ソケットをポートにバインドし、リスニングソケットを設定することにより、外部からの接続を受信します。accept がコールされますが、外部からの接続リクエストを受信するまで、ブロックされます。acceptが返る時には、sourceAddr構造体に外部からの接続リクエストのオリジナルIPアドレスと外部からの接続リクエストのポートが書き込まれます。acceptは、新規ソケットを作成し、相手から接続が閉じられるまでデータ受信に使用されます。この後、アプリケーションは、外部からの接続リクエストを待機する状態に戻ります。

```
#include <trsocket.h>

#define TM_BUF_SIZE 1400
#define TM_DEST_PORT 9999

char testBuffer[TM_BUF_SIZE];
char * strError;
```

```
int tcpServer(void)
{
    int                 listenSocket;
    int                 newSocket;
    struct sockaddr_in sourceAddr;
    struct sockaddr_in destAddr;
    int                 errorCode;
    int                 addrLen;
    int                 returnVal;

    returnVal = 0;

/* struct sockaddr_inのデータの長さを指定する。*/
    destAddr.sin_len = sizeof(sockaddr_in);

/* アドレスファミリーを指定する。*/
    destAddr.sin_family = AF_INET;

/* 送信先のポートを選択する。( これは、サーバーなので、送信先のポートはバインド先となる。) */
    destAddr.sin_port = htons(TM_DEST_PORT);

/*
 * 送信先のIPアドレスを選択する(IPアドレス) この値を0に設定すると、どんなI P アドレスを使用するか考慮せずに、ひとつ選ぶことになる。I P アドレスをひとつだけ持つシステムにとっては、もっとも簡単なアプローチである。
*/
    destAddr.sin_addr.s_addr = inet_addr("0.0.0.0");

/* ソケットを作成する。*/
    listenSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

/* 適切にソケットが作成されたか確認する。*/
    if (listenSocket == TM_SOCKET_ERROR)
    {
        returnVal = tfGetSocketError(listenSocket);
        errorStr = tfStrError(returnVal);
```

```
            goto TCPServerEnd;
        }

/* ソケットデータを受信したい場所で、ポートおよびアドレスにバインドする。 */
        errorCode = bind(listenSocket, &destAddr, sizeof(destAddr));

/* bind内のエラーを確認する。*/
        if (errorCode < 0)
        {
            returnVal = tfGetSocketError(listenSocket);
            errorStr = tfStrError(returnVal);
            goto TCPServerEnd;
        }

/* ソケットをリスニングソケットとして設定する。*/
        errorCode = listen(listenSocket, 10);

/* listen内のエラーを確認する。*/
        if (errorCode < 0)
        {
            returnVal = tfGetSocketError(listenSocket);
            errorStr = tfStrError(returnVal);
            goto TCPServerEnd;
        }

/* 継続する。*/
        while (1)
        {
        /* 構造体sockaddr_inのサイズを取得する。*/
            addrLen = sizeof(sourceAddr);
/*
 * 受信接続リクエストを受信する。ソースアドレス・ポートは、sourceAddrに格納される。sourceAddrに
書き込まれるデータの長さは、addrLenに格納される。sourceAddrに多数のバイトが書き込まれないよ
うにaddrLenの初期値がチェックされる。
*/
            newSocket = accept(listenSocket, &sourceAddr, &addrLen);
```

```
/* accept 内のエラーをチェックする。*/
        if (newSocket < 0)
        {
            returnVal = tfGetSocketError(listenSocket);
            errorStr = tfStrError(returnVal);
            goto tcpServerEnd;
        }
    /* 継続する*/
        while (1)
        {
    /* acceptによって作成された新規ソケット上のデータを受信する。*/
            errorCode = recv(newSocket, testBuffer, TM_BUF_SIZE, 0);
/* エラーがないか確認する。*/
            if (errorCode < 0)
            {
                tfClose(newSocket);
                returnVal = tfGetSocketError(newSocket);
                errorStr = tfStrError(returnVal);
                goto tcpServerEnd;
            }
    /* 0バイトデータ受信は、接続が閉じていることを示す。その場合、新規ソケットを閉じ、そのループ(the inner)を中止する。
    */
            if (errorCode == 0)
            {
                tfClose(newSocket);
                break;
            }
        }
    }
    tcpServerEnd:
/* 閉じる前にソケットがあるか確認する。*/
    if (listenSocket != -1)
    {
/* リッスニングソケットを閉じる。*/
        tfClose(listenSocket);
```

```
    }
    return(returnVal);
}
```

3. プログラマーリファレンス

BSD4.4ソケットAPI関数
accept
bind
connect
getpeername
getsockname
getsockopt
htonl
htons
inet_addr
inet_aton
inet_ntoa
listen
ntohl
ntohs
readv
recv
recvfrom
rresvport
select
send
sendto
shutdown
socket
tfClose
writev

ソケット拡張関数
tfGetSocketError

ARP/ルーティングテーブルAPI
tfAddDefaultGateway

KASAGO初期化関数
tfKasagoInitialize

デバイス/インタフェースAPI
tfCloseInterface
tfConfigInterface
tfGetIpAddress
tfOpenInterface
tfUnConfigInterface

コンパイラーライブラリ置換関数
tfStrError

PING API
tfPingClose
tfPingGetStatistics
tfPingOpenStart

BSD4.4 ソケット API

accept

include <trsocket.h>

```
int                     accept
(
    int                 socketDescriptor,
    struct sockaddr * addressPtr,
    int *               addressLengthPtr
);
```

関数の説明

引数socketDescriptorはsocketによって作成されるソケットで、bindによってアドレスを設定し、listenコール後に接続を待ちます。acceptは接続保留中のキューから最初の接続を抜き出し、socketDescriptorのプロパティを使用して、新規に接続済みソケットを作成し、新しいソケットディスクリプターを割り当てます。 接続のキューがなく、ソケットがブロッキングモードの場合、接続が発生するまでacceptコールはブロックします。ソケットがノンブロッキングモードで接続のキューがない場合、acceptはエラーを返します。

割り当てられたソケットは、接続されたソケットとの間のsendおよびrecvによるデータの送受信に使用されます。接続を受け入れる目的には使用できません。 最初のソケットは接続をさらに受け入れるために、開放した状態のままになります。 acceptは、接続指向のソケットタイプ(SOCK_STREAM)で使用されます。

selectを使用する(acceptをコールする前)

acceptを目的としてselectの読み取り条件を選択し、コールすることが出来ます。ただし、これは接続が保留中であることを示すだけなので、acceptをコールする必要があります。

パラメータ	説明
socketDescriptor	socketで作成され、bindで結び付けられ、listenで接続待ちにあるソケットディスクリプタ。
addressPtr	接続先のアドレスが書き込まれる構造体。
addressLengthPtr	addressPtr構造体の長さを設定する。返却時には実際の長さが格納される。

戻り値

新しいソケットディスクリプタ、またはエラーの場合は-1を返却します。
エラーの場合は、tfGetSocketError(socketDescriptor)で、以下のエラーコードを取得できます。

TM_EBADF	ソケットディスクリプタが無効。
TM_EINVAL	addressPtrがヌルポインタ。
TM_EINVAL	addressLengthPtrはヌルポインタ。
TM_EINVAL	addressLengthPtrの値が小さすぎる。
TM_ENOBUFS	オペレーションを完了するだけの使用可能ユーザメモリがない。
TM_EPERM	listenをコールする前に、acceptをコールできない。
TM_EOPNOTSUPP	指定されたソケットはSOCK_STREAMタイプではない。
TM_EWOULDBLOCK	ソケットがノンブロッキングになっていて、受け入れられる接続がない。

bind

```
#include <trsocket.h>

int                             bind
(
    int                          socketDescriptor,
    const struct sockaddr * addressPtr,
    int                          addressLength
);
```

関数の説明

bind はソケットにアドレスを割り当てるために使用します。TCP、UDPプロトコルではIPアドレスとポート番号がソケットアドレスになります。
通常、クライアント側はbindを呼び出さずに、プロトコルスタックによりランダムにポート番号を割り当ててもらいますが、bindを使用して特定のポート番号を割り当てることも出来ます。
また、サーバ側は、通常"well known"ポート番号をbindで割り当てます。ポート番号は65535まで指定できます。

パラメータ	説明
socketDescriptor	IPアドレスとポート番号を割り当てるソケットディスクリプタ。
addressPtr	割り当てるアドレスを含む構造体へのポインタ。
addressLength	アドレス構造体の長さ。

戻り値	意味
0	成功
-1	エラーが発生

bindは、以下の場合にエラーになります。

TM_EADDRINUSE	指定アドレスが使用中である。
TM_EAFNOSUPPORT	指定のアドレスファミリーはこのソケットでは使用できない。
TM_EBADF	ソケットディスクリプタが無効。
TM_EINVAL	パラメータの1つが無効か、ソケットがバインドされている。

connect

```
#include <trsocket.h>

int                             connect
(
    int                         socketDescriptor,
    const struct sockaddr * addressPtr,
    int                         addressLength
);
```

関数の説明

パラメータsocketDescriptorはソケットです。このタイプがSOCK_DGRAMの場合、本関数ではソケットに対応付けられる通信相手を指定します。このアドレスはデータグラムの送信先であり、データグラム受信時の送信元となる唯一のアドレスです。ソケットのタイプがSOCK_STREAMの場合は、通信相手のソケットへ接続を試みます。通信相手のソケットはaddressPtrにより指定されます。

addressPtrは、通信相手のソケットのポート番号とIPアドレスへのポインタです。socketDescriptorがバインドされていない場合、ソケットは下位のトランスポート層が選択したアドレスにバインドされます。通常、ストリームソケットがconnectに成功出来るのは一度だけです。データグラムソケットはconnectを複数回使用してその対応付けを変更出来ます。データグラムソケットは、ヌルアドレスに接続することにより対応付けを解除することが出来ます。

non-blocking connectが許可されていることに注意して下さい。この場合接続が完了するまで、connectコールはTM_EINPROGRESSエラーコードでエラーになります。さらに接続完了後はTM_EISCONN, クローズ開始後はTM_ESHUTDOWNのエラーを返却します。(R3.0からR4.1.2.49までconnectへの追加コールは、これらの場合TM_EALREADYを返却します。)

non-blocking connectとselect

もう一つの方法としてユーザは、接続が完了する時をチェックするために、そのソケットディスクリプタ用の書き込みマスクでconnect発行した後でselectをコールすることが出来ます。

non-blocking connect とtfRegisterSocketCB

もう一つの方法として、**connect** コール完了の非同期通知を受け取るため、ノンブロッキングで**connect**を発行する前に、TM_CB_CONNECT_COMPLTイベントフラグ付きのtfregisterSocketCBをコールすることが出来ます。

パラメータ	**説明**
socketDescriptor	ポート番号を割り当てるソケットディスクリプタ(bindされていない場合)
addressPtr	TCPの場合は、接続するアドレスを含む構造体へのポインタ。UDPの場合は、送信先のデフォルトアドレスであり、受け取り先の唯一のアドレス。
addressLength	アドレス構造体の長さ。

戻り値	**意味**
0	成功(ノンブロッキングでも即座に接続が完了したときに0が返却される場合があります)
-1	エラーが発生

connectは、以下の理由によりエラーになります。

TM_EADDRINUSE	ソケットアドレスが使用中。呼び出しプログラムはソケットディスクリプタを閉じ、再度、connectコールをする前にsocketをコールして新規ディスクリプタを取得する必要がある。
TM_EADDRNOTAVAIL	指定アドレスは通信相手として使用できない。
TM_EAFNOSUPPORT	指定のアドレスファミリーはこのソケットでは使用できない。
TM_EALREADY	ソケットはノンブロッキングであり、既にconnectが呼び出されている(R3.0～R4.1.2.49)。
TM_EBADF	ソケットディスクリプタが無効。
TM_ENOBUFS	オペレーションを完了するだけのメモリがない。
TM_ECONNREFUSED	接続試行が強制的に拒否された。呼び出しプログラムはソケットディスクリプタを閉じ、再度、connectコールをする前にsocketをコールして新規ディスクリプタを取得する必要がある。
TM_EPERM	listenコール後はconnectをコールすることは出来ない。
TM_EINVAL	パラメータの1つが無効。
TM_EISCONN	ソケットは接続完了済み。 （ノンブロッキングで最初のconnectコールでエラー復帰後、tfGetSocketErrorがコールされるまでの間に接続が完了した場合、tfGetSocketErrorの返却値は0になります）
TM_EHOSTUNREACH	接続先ホストへの接続経路がない。
TM_ETIMEDOUT	ソケットはブロッキングであり、接続が確立される前に接続確立のタイムアウトが発生した。呼び出しプログラムはソケットディスクリプタを閉じ、再度、connectコールをする前にsocket をコールして新規ディスクリプタを取得する必要がある。
TM_EINPROGRESS	ソケットはノンブロッキングであり、現在の接続試行がまだ完了していない。
TM_ESHUTDOWN	接続確立のタイムアウトが発生、または接続試行が強制的に拒否されたため、クローズ状態に移行した。またはtfCloseを呼び出し、クローズを開始した。(R4.1.2.50以降)

getpeername

```
#include <trsocket.h>

int                     getpeername
(
    int                 socketDescriptor,
    struct sockaddr * fromAddressPtr,
    int *               addressLengthPtr
);
```

関数の説明

この関数は、ソケット接続先であるリモートシステムのIPアドレス/ポート番号を呼び出し元に返します。

パラメータ	説明
socketDescriptor	情報の取得が必要なソケットディスクリプタ。
fromAddressPtr	この情報を挿入する必要があるアドレス構造体へのポインタ。
addressLengthPtr	アドレス構造体の長さ。

戻り値	意味
0	成功
-1	エラーが発生

getpeernameは以下の理由でエラーになります。

TM_EBADF	ソケットディスクリプタが無効。
TM_ENOTCONN	ソケットが未接続。
TM_EINVAL	パラメータの1つが無効。

getsockname

```
#include <trsocket.h>

int                         getsockname
(
    int                     socketDescriptor,
    struct sockaddr * myAddressPtr,
    int *                   addressLengthPtr
);
```

関数の説明

この関数は、指定ソケットで使用しているローカルIPアドレス/ポート番号を呼び出し元へ返します。

パラメータ	**説明**
socketDescriptor	問い合わせが必要なソケットディスクリプタ。
myAddressPtr	アドレス情報が格納されるアドレス構造体へのポインタ。
addressLengthPtr	アドレス構造体の長さ。

戻り値	意味
0	成功
-1	エラーが発生

getsocknameは、以下の場合エラーとなります。

TM_EBADF	ソケットディスクリプタが無効。
TM_EINVAL	パラメータの1つが無効。

getsockopt

```
#include <trsocket.h>

int         getsockopt
(
      int   socketDescriptor,
      int   protocolLevel,
      int   optionName,
      char * optionValuePtr,
      int   * optionLengthPtr
);
```

関数の説明

getsockoptは、ソケットに対応づけられたオプションを取得するために使用されます。オプションはさまざまなプロトコルレベルに関するものが存在しますが、オプションの提示は常に最上位の“ソケット”層で行われます。ソケットオプションを操作する時は、そのオプションが関連するプロトコルレベルとオプション名を指定します。ソケットレベルでオプションを操作する時は、protocolLevel はSOL_SOCKETに指定します。他のレベルでオプションを操作する場合は、protocolLevel はオプションを制御するプロトコルのプロトコル番号になります。例えば、オプションがTCPプロトコルで解釈されることを示すためには、protocolLevel にTCPのプロトコル番号を指定しなければなりません。getsockopt では、パラメータのoptionValuePtr とoptionLengthPtr に要求されたオプションの値が返却されるバッファを指定します。本関数コール時に、optionLengthPtr にはoptionValuePtr が示すバッファサイズを指定し、結果として実際に返却されるサイズの値に書き換えられます。インクルードファイル<trsocket.h>には、以下に記述されたオプションの定義が含まれています。オプションは書式と名前が異なります。ほとんどのソケットレベルオプションは、optionValuePtrにintタイプをとります。

SO_LINGER は必要なオプションの状態とリンガー間隔を指定するstruct lingerパラメータを使用します(以下参照)。struct linger は<trsocket.h >に定義されています。

struct linger は以下のメンバーを含みます。

l_onoff　on＝1／off＝0
l_linger　秒単位の継続時間

ソケットレベルでは以下のオプションが認識されます。

オプション	説明
SO_ACCEPTCONN	リスニングの状態。listenコールにより1になる。
SO_DONTROUTE	送信メッセージの経路指定バイパスを有効または無効にする。 デフォルトは0 （無効）
SO_KEEPALIVE	キープアライブを有効または無効にする。 デフォルトは0 (無効)
SO_LINGER	FINに対するAck受信までクローズ状態を継続する。 デフォルトは0 (無効)
SO_OOBINLINE	アウトオブバンドデータの受け取りを有効または無効にする。 デフォルトは0 （無効）
SO_REUSEADDR	異なったローカルIPアドレスを使用して、複数のソケットに同一ポート番号でbindすることを有効または無効にする。

	SO_REUSEADDRを有効にするには、trsystem.hにて TM_USE_REUSEADDR_LISTのマクロ定義が必要です。 デフォルトは0（無効）
SO_RCVLOWAT	受信のローウォーターマーク。
SO_SNDLOWAT	送信のローウォーターマーク。
SO_RCVBUF	受信用バッファサイズ。 デフォルトは8192バイト。
SO_SNDBUF	送信用バッファサイズ。 デフォルトは8192バイト。
TM_SO_RCVCOPY	TCPソケット：ソケット受信キュー内の前の受信バッファに空きがあるなら、今回受信したバッファのユーザデータをコピーして、受信バッファを解放する。 UDPソケット：ソケット受信キューに受信バッファがあるなら、今回受信したバッファのユーザデータ分の受信バッファを新たに取得しユーザデータをコピーして、受信バッファを解放する。 これは、事前に割り当てられた受信バッファのサイズが大きいまま受信キューに残らないようにするためのオプションである。ユーザデータサイズが取得した受信バッファサイズの1/4(25%)以下の場合に機能する。なおユーザが独自に受信バッファ領域を用意する場合は、本機能は無効となる。 デフォルト値は4（25%）
TM_SO_SNDAPPEND	TCPソケットのみ。TCP送信キューの前の送信バッファへの追加を試みる送信データサイズの閾値（バイト単位）。前のデータのACKを待っている状態で、次のsendの送信データサイズが閾値以下の場合に、TCP送信キュー内の前の送信バッファに空きがあるならユーザデータをコピーしてバッファを再編成する。これは、相手からのACKが遅延した場合、小さなデータを連続して送信するときにメモリを大量に消費することを回避するためのオプションである。 デフォルト値は128バイト。
TM_SO_RCV_DGRAMS	受信データグラムパケット(非TCP)キューの最大数。(R4.0.2.21以降) デフォルトは32（R4.1.2.29以前は8）
TM_SO_SND_DGRAMS	送信データグラムパケット(非TCP)キューの最大数。(R4.0.2.21以降) デフォルトは32（R4.1.2.29以前は8）

SO_REUSEADDRは、異なったローカルIPアドレス使用して、複数のソケットに同一ポート番号でbindコールすることを許可します。

SO_KEEPALIVEは、接続されたソケット上でキープアライブメッセージの周期的な送信(2時間ごと)を有効にします。接続先がこれらのメッセージに応答できない場合は、接続が解除されたと判断します。

SO_LINGERは、送信確認されていないデータがソケットに残っており、そしてクローズが発行されたときに行われる動作を制御します。SO_LINGERが設定されている場合、そのソケットにキューイングされたデータを全て送信完了するか（即ち，FINに対するACKを受信）、またはリンガータイマがタイムアウトになるまで、ソケットのクローズはブロックされます（SO_LINGER要求時、リンガー間隔と呼ばれるタイムアウト時間をsetsockoptで指定します）。SO_LINGERが無効で、ソケットのクローズが発行されると、ただちにソケットはクローズされます。

アウトオブバンドデータに対応するプロトコルを使用した場合、SO_OOBINLINEオプションを有効にすると、受信したアウトオブバンドデータが通常の受信キューに入ることを要求します。これで、MSG_OOBフラグがなくてもデータをrecvコールでアクセスすることが出来ます。

SO_SNDBUFとSO_RCVBUFは、それぞれ送信及び受信用のバッファに割り当てられたバッファサイ

ズを調整するオプションです。

IP レベルでは、以下のオプションが認識されます。

オプション	説明
IPO_MULTICAST_IF	マルチキャストデータグラム用に、設定されたインタフェースのIPアドレスを取得する。
IPO_MULTICAST_TTL	送信マルチキャストデータグラムのTTLのデフォルト値を取得する。 デフォルトは1
IPO_SRCADDR	自IPソースアドレスを取得する。
IPO_TOS	IPのサービスタイプ。 デフォルトは0
IPO_TTL	IPの生存時間。 デフォルトは64。

TCP レベルでは、以下のオプションが認識されます。

オプション	説明
TCP_KEEPALIVE	接続がアイドル状態になってから、キープアライブプローブを送信するまでの時間を秒単位で取得する。キープアライブプローブは、SO_KEEPALIVEソケットオプションにより有効にした場合にのみ送信されることに注意。 デフォルトは7200秒
TCP_MAXRT	相手が応答しないときに、TCPの再送を開始してから接続を中止するまでの時間を秒単位で取得する。値が 0 の場合はデフォルトを使用することを、また -1 の場合は永久に再送することを意味する。それ以外の値は、接続を中止するまでの時間を意味し、本タイマがタイムアウトするか、再送回数がTM_TCP_MAX_REXMITに達した場合に接続をあきらめる。後述の TM_TCP_MAX_REXMIT を参照。デフォルト動作は、接続確立前は RTT が不明のため 75秒、また接続確立後は、RTO でTM_TCP_MAX_REXMIT回の監視のみ。 デフォルトは 0。
TCP_MAXSEG	ネットワーク上で送信される最大TCPセグメントサイズを取得する。TCP_MAXSEG値は、セグメントごとに相手に送信できる最大データ量である(TCPオプションを含むが、TCPヘッダは含まない) 。つまり、セグメントあたりに送信されるユーザデータ量は、TCP_MAXSEGオプションで与えられる値マイナス使用可能なTCPオプションで与えられる値(例えば、TCPタイムスタンプオプションは12バイト)になる。 デフォルトはIP MTUマイナス40バイト
TCP_NODELAY	このオプション値が0以外の場合、TCPバッファ内で送信データをバッファリングするNagleアルゴリズムを無効にする。データ量が少ない場合でも、小パケットを出来るだけ早く送信させるためには有効である。オプション値が0の場合は、小さなデータを連続して送信してもデータはバッファリングされるため、小パケットを頻繁に送らずにすみ、ネットワークを有効に利用できる。 デフォルトは0
TCP_NOPUSH	このオプション値が0以外の場合、TCPバッファ内でフルサイズのセグメントがバッファリングされるまで、データの送信を遅延させる。遅

	延させることのできる時間の上限はプローブタイマ(後述のTM_TCP_PROBE_MIN参照)に依存する。この上限に達すると、バッファリングされたデータは自動的に送信される。 FTPのように、大量のデータを連続的に送信するアプリケーションに有効である。オプション値が0の場合は、TCPバッファを空にする場合に非フルサイズセグメントのデータを送信する 。 デフォルトは0
TCP_STDURG	このオプション値が0の場合、緊急ポインタの扱いがBSD互換の方法で行われ、緊急ポインタは緊急データの最後のバイト＋1を指す。 オプション値が1の場合は、RFC1122の記述に従い、緊急ポインタは緊急データの最後のバイトを指す。 デフォルトは1
TM_TCP_SEL_ACK	このオプション値が0以外の場合、TCP selective Acknowlegmentオプションは使用可能となる。 デフォルトは1
TM_TCP_WND_SCALE	このオプション値が0以外の場合、TCPウィンドウスケールオプションが使用可能。 デフォルトは1
TM_TCP_TS	このオプション値が0以外の場合、TCPタイムスタンプオプションが使用可能。 デフォルトは1
TM_TCP_SLOW_START	このオプション値が0以外の場合、TCPスロースタートアルゴリズムが使用可能となる。 デフォルトは1
TM_TCP_DELAY_ACK	TCP遅延ack時間をミリ秒単位で取得する。 デフォルトは200ミリ秒
TM_TCP_MAX_REXMIT	リモートから応答がない場合、TCP接続をあきらめる。最大再送信数を取得する。上述のTCP_MAXRTも参照。 デフォルトは12
TM_TCP_KEEPALIVE_CNT	リモートから応答がない場合、TCP接続をあきらめるキープアライブ最大再送信数を取得する。上述のTCP_KEEPALIVEを参照。 デフォルトは8
TM_TCP_FINWT2TIME	TCPがクローズを開始後、リモート側がクローズするのを待機する最大時間量を取得する。 デフォルトは600秒
TM_TCP_2MSLTIME	接続のクローズをいったん開始して、TCPがTIME WAIT 状態で待機する最大時間量を取得する。 デフォルトは60秒
TM_TCP_RTO_DEF	ラウンドトリップタイム(RTT)が計算されていないSYN/SYN ACKパケット，及び最初のデータパケット時に使用されるTCPデフォルト再送信タイムアウト値をミリ秒で取得する。(R3.32以前は接続確立時にRTTによる再送信タイムアウトの更新を行っていたため、この値が使用されるのはSYN/SYN ACKパケットのみ) デフォルトは3000ミリ秒
TM_TCP_RTO_MIN	最小再送信タイムアウトをミリ秒で取得する。RTTにより再計算される再送信タイムアウトはTM_TCP_RTO_MINとTM_TCP_RTO_MAXの間になる。 デフォルトは1000ミリ秒 (R4.1.2.45以前は100ミリ秒)
TM_TCP_RTO_MAX	最大再送信タイムアウトをミリ秒で取得する。RTTにより再計算される再送信タイムアウトはTM_TCP_RTO_MINとTM_TCP_RTO_MAXの間

	になる。 デフォルトは60000ミリ秒（R4.0.2.27以前は64000ミリ秒）
TM_TCP_PROBE_MIN	最小ウィンドウプローブタイムアウトをミリ秒で取得する。RTTにより再計算されるウィンドウプローブタイムアウトは、TM_TCP_PROBE_MINとTM_TCP_PROBE_MAXの間になる。 デフォルトは500ミリ秒
TM_TCP_PROBE_MAX	最大ウィンドウプローブタイムアウトをミリ秒で取得する。RTTにより再計算されるウィンドウプローブタイムアウトは、TM_TCP_PROBE_MINとTM_TCP_PROBE_MAXの間になる。 デフォルトは60000ミリ秒
TM_TCP_KEEPALIVE_INTV	キープアライブの再送間隔を秒単位で取得する。 TM_TCP_KEEPALIVE_CNTを参照。 デフォルトは75秒

パラメータ	**説明**
socketDescriptor	オプションを取得するソケットディスクリプタ。
protocolLevel	オプションを取得するプロトコル。 下記参照。
optionName	取得するオプション。 上記及び下記参照。
optionValuePtr	オプション値が返却されるユーザ変数へのポインタ。ユーザ変数のデータタイプは下記のとおり。
optionLengthPtr	ユーザ変数のサイズ。下記オプションデータタイプのサイズである。入出力両用の引数であり、コール前に設定し結果として実際に返却されるサイズの値に書き換えられる。

プロトコルレベル	**説明**
SOL_SOCKET	ソケットレベルプロトコル
IP_PROTOIP	IPレベルプロトコル
IP_PROTOTCP	TCPレベルプロトコル

プロトコルレベル	オプションネーム	オプションデータタイプ	オプション値
SOL_SOCKET	SO_ACCEPTCON	int	0 or 1
	SO_DONTROUTE	int	0 or 1
	SO_KEEPALIVE	int	0 or 1
	SO_LINGER	struct linger	
	SO_OOBINLINE	int	0 or 1
	SO_RCVBUF	unsigned long	
	SO_RCVLOWAT	unsigned long	
	SO_REUSEADDR	int	0 or 1
	SO_SNDBUF	unsigned long	
	SO_SNDLOWAT	unsigned long	
	TM_SO_RCVCOPY	unsigned int	
	TM_SO_SNDAPPEND	unsigned int	
	TM_SO_RCV_DGRAMS	unsigned long	
	TM_SO_SND_DGRAMS	unsigned long	
IP_PROTOIP	IPO_MULTICAST_IF	struct in_addr	
	IPO_MULTICAST_TTL	unsigned char	
	IPO_TOS	unsigned char	
	IPO_TTL	unsigned char	
	IPO_SRCADDR	ttUserIpAddress	
IP_PROTOTCP	TCP_KEEPALIVE	int	
	TCP_MAXRT	int	
	TCP_MAXSEG	int	
	TCP_NODELAY	int	0 or 1
	TCP_NOPUSH	int	0 or 1
	TCP_STDURG	int	0 or 1
	TM_TCP_2MSLTIME	int	
	TM_TCP_DELAY_ACK	int	
	TM_TCP_FINWT2TIME	int	
	TM_TCP_KEEPALIVE_CNT	int	
	TM_TCP_KEEPALIVE_INTV	int	
	TM_TCP_MAX_REXMIT	int	
	TM_TCP_PROBE_MAX	unsigned long	
	TM_TCP_PROBE_MIN	unsigned long	
	TM_TCP_RTO_DEF	unsigned long	
	TM_TCP_RTO_MAX	unsigned long	
	TM_TCP_RTO_MIN	unsigned long	
	TM_TCP_SEL_ACK	int	0 or 1
	TM_TCP_SLOW_START	int	0 or 1
	TM_TCP_TS	int	0 or 1
	TM_TCP_WND_SCALE	int	0 or 1

戻り値	意味
0	オプションの取得は成功
-1	エラーが発生

getsockoptは以下の場合エラーになります。

TM_EBADF	ソケットディスクリプタが無効
TM_EINVAL	パラメータの1つが無効
TM_ENOPROTOOPT	オプション名は指定のプロトコルレベルで不明
TM_EOPNOTSUPP	指定のオプション名は未サポート

htonl

```
#include <trsocket.h>

unsigned long        htonl
(
      unsigned long longValue
);
```

関数の説明

long値をホストバイト順からネットワークバイト順に変換します。

パラメータ	説明
longValue	変換する値

戻り値

変換された値

htons

```
#include <trsocket.h>

unsigned short      htons
(
      unsigned short shortValue
);
```

関数の説明

short値をホストバイト順からネットワークバイト順に変換します。

パラメータ	説明
shortValue	変換する値

戻り値

変換された値

inet_addr

```
#include <trsocket.h>

unsigned long inet_addr
(
      char * ipAddressDottedStringPtr
);
```

関数の説明

IP アドレスを小数点表記からバイナリフォーマットへ変換します。

パラメータ	説明
ipAddressDottedStringPtr	小数点付文字列(例"208.229.201.4")

戻り値	意味
-1	エラーが発生
その他	ネットワークバイト順のIPアドレス

inet_aton

```
#include <trsocket.h>

unsigned long inet_aton
(
    char * ipAddressDottedStringPtr
);
```

関数の説明

IP アドレスを小数点表記からバイナリフォーマットへ変換します。

パラメータ	説明
ipAddressDottedStringPtr	小数点付文字列 (例“208.229.201.4”)

戻り値	意味
0	エラーが発生
その他	ネットワークバイト順のIPアドレス

inet_ntoa

```
#include <trsocket.h>

char *              inet_ntoa
(
      struct in_addr inAddr
);
```

関数の説明

IPアドレス構成から(sockaddr_in構造体のsin_addr エレメント)小数点表記のASCII文字列へ変換します。

注意： inet_ntoaはリエントラント可能ではありません。BSDサポート用にのみ用意されています。

パラメータ	説明
inAddr	変換するアドレスを含む構造体

戻り値	意味
0	エラーが発生
その他	小数点表記のIPアドレスのACII文字列へのポインタ

listen

```
#include <trsocket.h>

int      listen
(
     int socketDescriptor,
     int backLog
);
```

関数の説明

接続を受け入れるため、ソケットはまずsocketで作成され、接続要求の受信に対するbackLogがlistenで指定され、接続はacceptで受け入れられます。listenコールは、SOCK_STREAMタイプのソケットのみを使用します。backLogパラメータは、接続保留中キューの最大個数を定義します。キューがいっぱいの状態で接続要求が到着すると、その要求を無視します(R4.1.2.49以前はRSTを送信し拒否)。この場合、通信相手のTCPは接続を再試行し、タイムアウトになるまで接続保留中のキューから抜き出さない場合、connectはTM_ETIMEDOUTでエラーとなります。

パラメータ	説明
socketDescriptor	listenを有効にするソケットディスクリプタ
backLog	ソケットに許可された接続保留中キューの最大個数

戻り値	意味
0	成功
-1	エラーが発生

listen は以下の場合エラーとなります。

TM_EADDRINUSE	ソケットが現在別の接続に使用されている。
TM_EBADF	ソケットディスクリプタが無効。
TM_EINVAL	パラメータの1つが無効。
TM_EOPNOTSUPP	ソケットのタイプがlistenオペレーションに対応していない。

ntohl

```
#include <trsocket.h>

unsigned long        ntohl
(
      unsigned long longValue
);
```

関数の説明

long値をネットワークバイト順からホストバイト順に変換します。

パラメータ	説明
longValue	変換する値

戻り値

変換された値

ntohs

```
#include <trsocket.h>

unsigned short        ntohs
(
      unsigned short shortValue
);
```

関数の説明

short値をネットワークバイト順からホストバイト順に変換します。

パラメータ	説明
shortValue	変換する値

戻り値

変換された値

readv

```
#include <trsocket.h>

int                 readv
(
    int             socketDescriptor,
    struct iovec * iov,
    int             iovcnt
);
```

関数の説明

readvは分散読み込みとして機能します。すなわち受信したデータを複数のバッファに置くことが可能です。readvはsocketDescriptorからiovcnt個のバッファ分のデータを読み込み、iovで指定されたバッファに格納します。

iov[0], iov[1], ..., iov[iovcnt-1].

iovec構造体には以下のメンバーが含まれます。

```
caddr_t iov_base;
int iov_len;
```

各iovecエントリは、データが格納されるベースアドレスとエリアの長さを指定します。
readvは次のバッファへ進む前に最初のバッファをいっぱいにします。成功した場合、readvはバッファの中に実際に読み込まれて格納されたバイト数を返却します。ただし、この数は全てのiov_len値の合計より少ないこともあります。EOF（ファイルの終わり）に達すると0の値が返されます。

パラメータ	説明
socketDescriptor	データの受信元になるソケットディスクリプタ
iov	受信データ格納先のバッファリスト
iovcnt	リスト内のバッファ数

戻り値	意味
>0	実際にソケットから読み込まれたバイト数
0	EOF
-1	エラーが発生

readv は以下の場合にエラーとなります。

TM_EBADF	ソケットディスクリプタが無効。
TM_EINVAL	iovcntが0以下。iov_lenの値の合計が整数をオーバーフローした。
TM_ENOBUFS	オペレーションを完了するだけのメモリがない。
TM_EMSGSIZE	受信バッファが小さすぎるため、バッファを超過したデータを破棄した。
TM_EWOULDBLOCK	ソケットがノンブロッキングになっており、読み込み可能なデータがない。
TM_ECONNRESET	接続が相手側によって拒否された(TCPソケットのみ)。
TM_ESHUTDOWN	ユーザが読み込みshutdown、またはtfCloseを呼び出した(TCPソケ

	ットのみ)。
TM_ENOTCONN	ソケットが接続されていない。または接続が相手側によって拒否された、あるいは自ら拒否したため、クローズ状態に移行した。
TM_ETIMEDOUT	再送オーバが発生した。
TM_EHOSTUNREACH	宛先ホストへの経路がない。

recv

```
#include <trsocket.h>

int             recv
(
      int       socketDescriptor,
      char * bufferPtr,
      int       bufferLength,
      int       flags
);
```

関数の説明

recvは、他のソケットからメッセージを受信するために使用されます。recvは、接続されたソケット(connect及びacceptを参照)でのみ使用出来ます。ソケットディスクリプタは、socketまたはacceptで作成されるソケットです。この関数の戻り値として、受信したメッセージの長さが返却されます。メッセージが、与えられたバッファに対して長すぎる場合、メッセージの受信元のソケットタイプによっては、バッファを超過したデータが破棄されることがあります(socket参照)。返却されたメッセージの長さが、bufferLengthより小さくなることもあります(これはエラーではありません)。ソケットにメッセージがない場合、recvコールはメッセージの到着を待ちます。ただし、ソケットがノンブロッキングの場合、またはMSG_DONTWAITフラグがセットされている場合はメッセージの到着を待たず、ソケットエラーがTM_EWOULDBLOCKにセットされた状態で-1が返却されます。

ストリームにないアウトオブバンドデータ(SO_OOBINLINEオプションが設定されていない時(デフォルト)の緊急データ) (TCP プロトコルのみ)

SO_OOBINLINEオプションが設定されていない場合、TCPプロトコルには、単一のアウトオブバンドデータが渡されます。1つのアウトオブバンドデータがある場合、MSG_OOBフラグが設定されていないrecv要求では、アウトオブバンドデータの位置を超えて読み取ることは出来ません。つまり、現在の読み取り位置からアウトオブバンドバイトまで10バイトあり、20バイトのbufferLengthを指定し、フラグ値を0に指定してrecvを実行した場合、recvは10バイトのみを返します。この強制停止機能により、アウトオブバンドバイトマークに到達したことを判断するために、SOIOCATMARK要求で tfIoctlを実行することが出来ます。あるいは、もう一つの方法として、アウトオブバンドデータバイトのオフセットを決定するためにtfIoctlの代わりにtfGetOobDataOffsetを使うこともできます。マークに到達すると、MSG_OOBフラグが設定されたrecvは、アウトオブバンドデータバイトを読み取ることが出来ます。アウトオブバンドデータが到着した時、またはいつ到着するのか知るために、selectかtfRegisterSocketCBを使う必要があることに注意して下さい。

selectは、アウトオブバンドデータがいつ到着か，及びそれ以降のデータがいつ到着したかを判断するために使用されます。

パラメータ	説明
socketDescriptor	データの受信元になるソケットディスクリプタ
bufferPtr	受信データ格納先のバッファ
bufferLength	bufferPtr が示すバッファエリアの長さ
Flags	以下参照 flagsパラメータは、1つまたは、or指定により複数設定できます。 MSG_DONTWAIT　　データ受信完了を待たず、即時復帰する。

MSG_OOB	通常の“インバンド”データでなく、ソケットに提示される“アウトオブバンド”データを読み込む。
MSG_PEEK	受信処理の際に、受信キューからデータを削除しない。よって、この後でもう一度受信コールを呼び出すと、同じデータが返ることになる。

戻り値	**意味**
>0	ソケットから実際に受信されたバイト数
0	EOF
-1	エラーが発生

recvは以下の場合にエラーとなります。

TM_EBADF	ソケットディスクリプタが無効。
TM_ENOBUFS	オペレーションを完了するだけのメモリがない。
TM_EMSGSIZE	bufferLengthが小さすぎるため、バッファを超過したデータを破棄した。
TM_EWOULDBLOCK	ソケットがノンブロッキングになっており、読み込み可能なデータがない
TM_ECONNRESET	接続が相手側によって拒否された(TCPソケットのみ)。
TM_ESHUTDOWN	ユーザが読み込みshutdown、またはtfCloseを呼び出した(TCPソケットのみ)。
TM_EINVAL	パラメータの１つが無効か、MSG_OOBフラグを設定した場合に受信するアウトオブバンドデータがない。またはSO_OOBINLINEオプションを有効にしてMSG_OOBフラグを設定した。
TM_ENOTCONN	ソケットが接続されていない。または接続が相手側によって拒否された、あるいは自ら拒否したため、クローズ状態に移行した。
TM_ETIMEDOUT	再送オーバが発生した。
TM_EHOSTUNREACH	宛先ホストへの経路がない。

recvfrom

```
#include <trsocket.h>

int                     recvfrom
(
    int                  socketDescriptor,
    char *               bufferPtr,
    int                  bufferLength,
    int                  flags,
    struct sockaddr * fromPtr,
    int *                fromLengthPtr
);
```

関数の説明

recvfromは、他のソケットからメッセージを受信するために使用されます。recvfromを使用するとTCP以外のソケットで、接続済み, または未接続の何れのソケットを指定しても、データを受け取ることが出来ます。ソケットディスクリプタはsocket で作成するソケットです。fromPtrがNULLポインタでない場合、メッセージのソースアドレスが格納されます。fromLengthPtrは、入出力両用の引数です。このパラメータにfromPtrが示すバッファサイズを設定し、結果として実際に格納されたアドレスのサイズに書き換えられます。この関数の戻り値には、受信したメッセージの長さが返却されます。メッセージが、与えられたバッファに対して長すぎる場合、バッファを超過したデータが破棄されることがあります(socketを参照)。ソケットにメッセージがない場合、受信コールはメッセージが到着するのを待ちます。ただし、ソケットがノンブロッキングの場合、またはMSG_DONTWAITフラグがセットされている場合は、メッセージの到着を待たず、ソケットエラーがTM_EWOULDBLOCKにセットされた状態で-1が返却されます。

selectは、データがいつ到着したかを判断するために使用されます。

tfRegisterSocketCBは、非同期でデータがいつ到着したかを判断するために使用されます。

パラメータ	説明
socketDescriptor	データの受信元になるソケットディスクリプタ
bufferPtr	受信データ格納先のバッファ
bufferLength	bufferPtrが示すバッファエリアの長さ
flags	以下参照 flagsパラメータは1つまたはor指定により複数設定できます。 MSG_DONTWAIT　データ受信完了を待たず、即時復帰する。 MSG_PEEK　受信処理の際に、受信キューからデータを削除しない。よって、この後でもう一度受信コールを呼び出すと、同じデータが返ることになる。
fromPtr	送信元のアドレスが書き込まれる構造体。
fromLengthPtr	fromPtr構造体の長さを設定する。 返却時には実際の長さが格納される。

戻り値	意味
>0	実際にソケットから受信されたバイト数
0	EOF
-1	エラーが発生

recvfrom は、以下の場合にエラーになります。

TM_EBADF	ソケットディスクリプタが無効。
TM_EINVAL	パラメータの１つが無効。
TM_EMSGSIZE	bufferLengthが小さすぎるため、バッファを超過したデータを破棄した
TM_EPROTOTYPE	TCPプロトコルはrecvfromではなく、recvの使用を要求する。
TM_ENOBUFS	オペレーションを完了するだけのメモリがない。
TM_EWOULDBLOCK	ソケットがノンブロッキングになっており、読み込み可能なデータがない。

rresvport

```
#include <trsocket.h>

int         rresvport
(
    int * portToReservePtr
);
```

関数の説明

rresvportはTCPソケットを作成し、予約したポートをユーザが指定する予約ポートで始まるソケットにバインドするために使用されます。portToReservePtrパラメータは、入出力両用の引数です。portToReservePtrが指す整数は、この関数がバインドしようとする最初のポート番号です。呼び出し側は一般的に開始ポート番号をIPPORT_RESERVED-1に初期化します。(IPPORT_RESERVEDは1024に定義)。ポート番号がすでに使用されているためにバインドが失敗した場合、rresvportはポート番号を減少させ、再試行します。最終的にIPPORT_RESERVEDSTART(600に定義)に到達し、それが使用中であることがわかった場合、-1を返して、ソケットエラーはTM_EAGAINにセットされます。この関数が予約ポート番号のバインドに成功すると、ソケットディスクリプタをユーザに返し、ソケットがバインドされた予約ポートをportToReservePtrが指す整数セルに格納します。

パラメータ	説明
portToReservePtr	予約するポート番号と予約が成功したポート番号へのポインタ

戻り値	意味
>=0	ソケットディスクリプタ
-1	エラーが発生

エラーが発生した場合、ソケットエラーを取り出すにはtfGetSocketErrorを呼び出し、ソケットディスクリプタのパラメータにTM_SOCKET_ERRORを使用します。

rresvport は以下の場合にエラーとなります。

TM_EAGAIN	TCP/IPスタックは、IPPORT_RESERVEDSTARTと予約するポート番号との間で有効なポート番号を見つけることが出来ない。
TM_EINVAL	パラメータの1つが無効。ポインタがヌルか予約するポート番号がIPPORT_RESERVEDSTART(600)以下である。
TM_ENOBUFS	オペレーションを完了するだけのメモリがない。
TM_EMFILE	使用可能なソケットがこれ以上ない。

select

```
#include <trsocket.h>

int                     select
(
    int                 numberSockets,
    fd_set *            readSocketsPtr,
    fd_set *            writeSocketsPtr,
    fd_set *            exceptionSocketsPtr,
    struct timeval * timeOutPtr
);
```

関数の説明

select は、readSocketsPtr、writeSocketsPtr及びexceptionSocketsPtrなどでアドレスが渡されるソケットディスクリプタセットを調べ、それぞれのソケットディスクリプタが読み取り可能か、または書き込み可能か、または例外条件で中断しているかなどを確認します。アウトオブバンドデータは、唯一の例外条件です。numberSockets引数は、テストするソケットディスクリプタの数を指定します。その値は、テストする最大のソケットディスクリプタに1を加えたものです。ソケットディスクリプタセットの0から numberSockets－１までのソケットディスクリプタを調べます。selectは値を返す際、特定のソケットディスクリプタセットをレディ状態のソケットディスクリプタから構成されるサブセットと置き換えます。select へのコールでの戻り値は、レディソケットディスクリプタの数です。ソケットディスクリプタ設定は、ビットフィールドとして整数の配列に格納されます。以下のマクロはそのようなファイルディスクリプタセットを加工するために用意されています。

FD_ZERO(&fdset);	ソケットディスクリプタセット(fdset)をNULLに初期化する。
FD_SET(fd, &fdset);	特定のソケットディスクリプタfd を設定する。
FD_CLR(fd, &fdset);	fdsetからfd を削除する。
FD_ISSET(fd, &fdset);	fdが1ならば0以外、そうでなければ0を返却する。

selectはBSD Unix上ではファイルシステムとソケット両方に使用されているため、用語“fd”はBSD互換のために使用されます。

パラメータ	説明
numberSockets	テストするソケットディスクリプタの最大数+1
readSocketsPtr	読み取り条件をチェックするソケットマスクへのポインタ
writeSocketsPtr	書き込み条件をチェックするソケットマスクへのポインタ
exceptionSocketsPtr	例外条件（アウトオブバンドデータ）をチェックするソケットマスクへのポインタ
timeOutPtr	イベントを待機する時間長を含む構造体へのポインタ

本関数が終了するまでの最大待ち時間を下記構造体のメンバへ設定します

```
struct        timeval
{
      u_long tv_sec; /*秒*/
      u_long tv_usec; /*マイクロ秒*/
}
```

上記構造体のポインタがNULLポインタの場合は指定した事象が発生するまで待ちとなります。構造体メンバへ0以外の数値を指定した場合は最大待ち時間の指定となります。構造体メンバに0を指定した場合は即時復帰となります。

戻り値	**意味**
>0	レディ状態のソケット数
0	タイムリミットを超過
-1	エラーが発生

エラーが発生した場合、ソケットエラーを取り出すには、tfGetSocketErrorを呼び出し、TM_SOCKET_ERRORをソケットディスクリプタパラメータとして使用します。selectは以下の場合、エラーとなります。

TM_EBADF	ソケットディスクリプタが無効。
TM_ENOBUFS	オペレーションを完了するだけのメモリがない。

send

```
#include <trsocket.h>

int         send
(
    int     socketDescriptor,
    char * bufferPtr,
    int     bufferLength,
    int     flags
);
```

関数の説明

sendは、メッセージを他の転送先へ送信するために使用されます。sendは、ソケットが接続状態の時のみ使用出来ます。ソケットディスクリプタは、socketまたはacceptで作成されたソケットです。
メッセージが長すぎて基本的なプロトコル(非TCPプロトコル)を使って分割して渡せない場合、ソケットエラーがTM_EMSGSIZEにセットされ、-1が返却されてメッセージは送信されません。-1が返却された場合は、エラーがローカルで検出されたことを示します。正の値が返却された場合、メッセージが届いたことを意味するわけではなく、メッセージが送信されたことを意味します。

ブロッキングソケット送信

ソケットに、送信するメッセージを保持するための十分な使用可能バッファスペースがない場合、sendはブロックします。

ノンブロッキングストリーム(TCP)ソケット送信

ソケットに、送信するメッセージを保持するための十分な使用可能バッファスペースがない場合、sendコールはブロックしません。メッセージからTCPバッファに許容するだけのデータが送信され、送信データの長さが返却されます。メッセージデータがひとつも受け入れられない場合、ソケットエラーがTM_EWOULDBLOCKにセットされ、-1が返却されます。

ノンブロッキングデータグラム送信

ソケットに、送信するメッセージを保持するだけの十分な使用可能バッファスペースがない場合、データは送信されず、ソケットエラーがTM_EWOULDBLOCKにセットされ-1が返却されます。

select関数は、より多くのデータを送信できるタイミングの特定に使用します。

パラメータ	説明
socketDescriptor	データ送信に使用するソケットディスクリプタ
bufferPtr	送信するバッファ
bufferLength	送信するバッファの長さ
flags	以下参照

flagsパラメータは1つまたはor指定により複数設定できます。

MSG_DONTWAIT	データ送信が完了するのを待たずに即時復帰する。
MSG_OOB	この概念に対応するソケットで“アウトオブバンド”データを送信する。AF_INETアドレスファミリーで作成されたSOCK_STREAMソケットだけがアウトオブバンドデータに対応する。
MSG_DONTROUTE	このオプションを使用するのは、診断またはルーティングプログラムのみ。

戻り値	意味
>=0	ソケット上で実際に送信されたバイト数
-1	エラーが発生

sendは、以下の場合にエラーになります。

TM_EBADF	ソケットディスクリプタが無効。
TM_EINVAL	パラメータの1つが無効。
TM_ENOBUFS	オペレーションを完了するだけのメモリがない。
TM_EHOSTUNREACH	非TCPソケットのみ。宛先ホストへの経路がない。
TM_EMSGSIZE	送信要求のメッセージが長過ぎる。
TM_EWOULDBLOCK	ソケットはノンブロッキングになっており、送信メッセージを保持するためのバッファスペースがない。
TM_ETIMEDOUT	再送オーバが発生した。
TM_ENOTCONN	ソケットが接続されていない。
TM_ECONNRESET	接続が相手側によって拒否された(TCPソケットのみ)。
TM_ESHUTDOWN	ユーザが書き込みshutdown、またはtfCloseを呼び出した。または接続が相手側によって拒否された、あるいは自ら拒否したため、クローズ状態に移行した。(TCPソケットのみ)。

sendto

```
#include <trsocket.h>

int                             sendto
(
    int                         socketDescriptor,
    char *                      bufferPtr,
    int                         bufferLength,
    int                         flags,
    const struct sockaddr * toPtr,
    int                         toLength
);
```

関数の説明

sendtoは、他の転送先へメッセージを送信するために使用されます。sendtoは接続済み，または未接続の何れのソケットでも使用出来ますが、TCPソケットには使用できません。socketDescriptorは、socket で作成されるソケットです。ターゲットのアドレスはtoPtrで、サイズはtoLengthで指定します。

メッセージが長すぎて分割して渡すことができない場合、ソケットエラーがTM_EMSGSIZEにセットされ、-1が返却されてメッセージは転送されません。

-1が返却された場合は、エラーがローカルで検出されことを示します。正の値が返却された場合はメッセージが届いたことを意味するわけではなく、メッセージが送信されたことを意味します。

ソケットに送信するメッセージを保持するだけのバッファスペースがない時は、sendtoはブロックします。ただし、ソケットがノンブロッキングモードか、MSG_DONTWAITフラグがセットされている場合は、ソケットエラーがTM_EWOULDBLOCKにセットされた状態で-1が返却されます。

select関数は、より多くのデータを送信できるタイミングを特定するのに使用します。

パラメータ	説明	
socketDescriptor	データ送信に使用するソケットディスクリプタ	
bufferPtr	送信するバッファ	
bufferLength	送信するバッファの長さ	
toPtr	データ送信先のアドレス	
toLength	toPtrがポイントする領域の長さ	
flags	以下参照	
	flagsパラメータは、1つまたはor指定により複数設定できます。	
	MSG_DONTWAIT	データ送信が完了するのを待たず、即時復帰する。
	MSG_DONTROUTE	このオプションを使用するのは、診断またはルーティングプログラムのみ。

戻り値	意味
>=0	ソケット上で実際に送信されたバイト数
-1	エラーが発生

sendtoは以下の場合エラーになります。

TM_EBADF	ソケットディスクリプタが無効。
TM_EINVAL	パラメータの1つが無効。
TM_EHOSTDOWN	相手ホストがダウンしている。

TM_ENOBUFS	オペレーションを完了するだけの使用可能なメモリがない。
TM_EAFNOSUPPORT	指定のアドレスファミリーはこのソケットでは使用できない。
TM_EHOSTUNREACH	宛先ホストへの経路がない。
TM_EMSGSIZE	送信要求のメッセージが長すぎる。
TM_EPROTOTYPE	TCPプロトコルは、sendtoではなくsendの使用を要求している。
TM_EISCONN	ソケットは接続済みで、sendtoではなくsendの使用を要求している。
TM_EWOULDBLOCK	ソケットがノンブロッキングになっており、送信メッセージを保持するためのバッファスペースがない。

shutdown

```
#include <trsocket.h>

int     shutdown
(
    int socketDescriptor,
    int howToShutdown
);
```

関数の説明

パラメータhowToShutdownが指定する読み込み、書き込み、その両方のいずれかをシャットダウンします。

パラメータ	説明
socketDescriptor	シャットダウンするソケット
howToShutdown	方向: 0 = 読み込み 1 = 書き込み 2 = 両方

戻り値	意味
0	成功
-1	エラーが発生

shutdown は以下の場合にエラーとなります。

TM_EBADF	ソケットディスクリプタが無効。
TM_EINVAL	パラメータの1つが無効。
TM_EOPNOTSUPP	指定されたソケットはSOCK_STREAMタイプではない。
TM_ESHUTDOWN	ソケットは既にクローズされている。

socket

```
#include <trsocket.h>

int         socket
(
        int family,
        int type,
        int protocol
);
```

関数の説明

socketは通信のための終点(endpoint)を作成し、ソケットディスクリプタを返します。familyパラメータは、通信が行われるドメインを指定し、この指定で使用すべきプロトコルファミリーを選択します。プロトコルファミリーは一般的に、その後にソケットで行われるオペレーションで指定されるアドレスのアドレスファミリーと同じです。これらのファミリーはインクルードファイル<trsocket.h>に定義されています。protocolにゼロを指定した場合は、ファミリーとタイプを含む最初のエントリーが使用されます。現在指定できるプロトコルファミリーは、ARPAインターネットプロトコルを示すPF_INETです。typeパラメータには、通信方式を定めるためにソケットのタイプを指定します。

現在、定義されているタイプ

SOCK_STREAM
SOCK_DGRAM
SOCK_RAW

SOCK_STREAMタイプは、連続した信頼性のある双方向接続ベースのバイトストリームを提供します。また、アウトオブバンドデータ送信メカニズムが提供されています。SOCK_DGRAMソケットは、データグラム(固定最大長で接続がなく信頼性の低いメッセージ)をサポートします。 SOCK_DGRAMユーザはrecv(recvfrom)コールで、その都度パケット全体を読み込む必要があります。受信データ格納先のバッファサイズが足りない場合は、超過したデータが破棄されTM_EMSGSIZEのエラーコードが返却されます。

protocolは、ソケットで使用する特定のプロトコルを指定します。通常、指定されたプロトコルファミリー内の特定のソケットタイプに対応するプロトコルはひとつだけです。ただし、複数のプロトコルが存在することもあり、この場合はある特定のプロトコルを指定する必要があります。使用するプロトコル番号は、通信が行われる“通信ドメイン”特有なものです。呼び出し側がプロトコルを指定する場合、そのプロトコルはソケットレベルのオプション要求にパッケージされ、下位のプロトコルレイヤへ渡されます。

タイプがSOCK_STREAMのソケットは全二重バイトストリームです。ストリームソケットは、そこでデータの送受信を行う前に、接続状態になっていなければなりません。他のソケットへの接続は、クライアント側のconnectで行われます。サーバ側ではlistenをコールした後に、acceptをコールしなければなりません。接続状態になったら、recvおよびsendコールを使用してデータを転送できます。セッションが完了したら、ソケットのクローズが必要です。

SOCK_STREAMを使用した通信プロトコルは、データの喪失や重複が起こらないことを保証します。もし送信バッファ内にあるデータを、一定の時間内に転送出来ない場合は、接続が切れていると判断され、返却値はエラー値(-1)が返され、ソケットエラーにTM_ETIMEDOUTがセットされます。TCPプロトコルはオプションとして、通信がない場合におよそ2時間ごとに強制的に送信を行わせることで、ソケットをウォームアップ状態に保つことができます。その後一定時間(例えば5分間)にわたって、アイドル状態になっているコネクション上で応答が得られない場合はエラーが通知されます。

SOCK_DGRAMとSOCK_RAWソケットは、sendtoコールで指定した通信相手にデータグラムを送信で

きます。データグラムは、通常recvfromで受信され、次のデータグラムとその送信元アドレスをともに返却します。ソケットのオペレーションは、ソケットレベルオプションにより制御されます。このオプションはファイル<trsocket.h>に定義されています。setsockoptとgetsockoptは、オプションの設定と取得に使用されます。

パラメータ	説明
family	このソケットに使用するプロトコルファミリー (現在、PF_INETだけが使用可能)
type	ソケットタイプ
protocol	このソケットに使用するレイヤー4のプロトコル

family	type	protocol	実際のプロトコル
PF_INET	SOCK_DGRAM	IPPROTO_UDP	UDP
PF_INET	SOCK_STREAM	IPPROTO_TCP	TCP
PF_INET	SOCK_RAW	IPPROTO_ICMP	ICMP
PF_INET	SOCK_RAW	IPPROTO_IGMP	IGMP

戻り値

新規ソケットディスクリプタ。エラー発生時は-1を返却する。エラーが発生した場合、tfGetSocketErrorをコールし、ソケットディスクリプタパラメータにTM_SOCKET_ERRORを使用することで、ソケットエラーを取り出せます。

socketは以下の場合にエラーとなります。

TM_EMFILE	使用可能なソケットがこれ以上ない。
TM_ENOBUFS	オペレーションを完了するだけのユーザメモリがない。
TM_EAFNOSUPPORT	指定されたファミリーは対応していない。
TM_EPROTONOSUPPORT	このファミリー内では、プロトコルタイプまたは指定されたプロトコルは対応していない。

tfClose

```
#include <trsocket.h>

int         tfClose
(
      int socketDescriptor
);
```

関数の説明

この関数はソケットをクローズするために使用されます。組み込みカーネルファイルシステムの関数と混同しないようにcloseと区別しています。

パラメータ	説明
socketDescriptor	クローズするソケットディスクリプタ

戻り値	意味
0	成功
-1	エラーが発生

tfCloseは以下の場合エラーとなります。

TM_EBADF	ソケットディスクリプタが無効
TM_EALREADY	前回のtfCloseコールが完了していない。
TM_ETIMEDOUT	タイムアウト値が0以外でリンガーオプションが有効の場合、リモートホストとのTCPクローズのハンドシェイクが完了する前にlingerタイムアウトが時間切れになった(ブロッキングTCPソケットのみ)。

writev

```
#include <trsocket.h>

int                     writev
(
int                     socketDescriptor,
const struct iovec * iov,
int                     iovcnt
);
```

関数の説明

writevは、分散書き込みとして機能します。すなわち送信するデータを複数のバッファに置くことが可能です。writevは、iov配列のメンバにより指定されるiovcnt個のバッファからデータを集め、ソケットディスクリプタに書き込みます。

iov[0], iov[1], ..., iov[iovcnt-1].

iovec構造体には以下のメンバーが含まれています。

```
caddr_t iov_base;
int iov_len;
```

各iovecエントリは、データの収集元であるバッファ領域のベースアドレスと長さを指定します。
writevは常に、次のバッファへ進む前に最初のバッファを完全に読み込みます。writevが成功すると、実際に書き込まれたバイト数が返却されます。このバイト数は、送信キューに十分なスペースがない場合、すべてのiov_len値の合計より少ないこともあります。

パラメータ	説明
socketDescriptor	データを書き込むためのソケットディスクリプタ
iov	データの収集、および送信元になるバッファリスト
iovcnt	リスト内のバッファ数

戻り値	意味
>=0	実際に書き込まれたバイト数
-1	エラーが発生

writev は以下の場合にエラーとなります。

TM_EBADF	ソケットディスクリプタが無効。
TM_EINVAL	iovcntが0以下。iov_len値の合計が整数をオーバーフローした。
TM_ENOBUFS	オペレーションを完了するだけのメモリがない。
TM_EMSGSIZE	送信要求のメッセージが長過ぎた。
TM_ECONNRESET	接続が相手側によって拒否された(TCPソケットのみ)。
TM_EWOULDBLOCK	ソケットがノンブロッキングになっており、送信メッセージを保持するためのバッファスペースがない。

TM_ENOTCONN	ソケットが接続されていない。
TM_ETIMEDOUT	再送オーバが発生した。
TM_ESHUTDOWN	ユーザが書き込みshutdown、またはtfCloseを呼び出した。または接続が相手側によって拒否された、あるいは自ら拒否したため、クローズ状態に移行した。(TCPソケットのみ)。
TM_EHOSTUNREACH	非TCPソケットのみ。宛先ホストへの経路がない。

ソケット拡張関数

tfGetSocketError

```
#include <trsocket.h>

int         tfGetSocketError
(
        int socketDescriptor
);
```

関数の説明

この関数は、ソケットコールが失敗した時に(TM_SOCKET_ERROR)、ソケットエラーを取得するために使用されます。

パラメータ	説明
socketDescriptor	エラーを取得するソケットディスクリプタ

戻り値	意味
ソケットの最終エラー値	

デバイス/インタフェース API

tfCloseInterface

```
#include <trsocket.h>

int                     tfCloseInterface
(
    ttUserInterface interfaceHandle
);
```

関数の説明

この関数は、オープン中のインタフェースをクローズするために使用されます。この関数では、ローカルルーティングテーブルからインタフェースを削除し、リンクレイヤのクローズ処理を行って、デバイスドライバクローズ関数を呼び出します。そしてリンクレイヤかデバイスドライバが返すエラーコードを返却します。

パラメータ	説明
interfaceHandle	クローズを行うインタフェースハンドル

戻り値	意味
0	成功
TM_EINVAL	パラメータの1つが無効。
TM_EINPROGRESS	クローズ処理中(PPP の場合)。ユーザは何もする必要はなく、インタフェースが実際にクローズした時に、PPP リンクレイヤからクローズ完了のイベントが通知されます。
その他	デバイスドライバクローズ関数で返却されたエラー値。

tfConfigInterface

```
#include<trsocket.h>

int                     tfConfigInterface
(
    ttUserInterface interfaceHandle,
    ttUserIpAddress ipAddress,
    ttUserIpAddress netMask,
    int             flag,
    int             buffersPerFrameCount,
    unsigned char   multiHomeIndex
);
```

関数の説明

この関数は、IPアドレス、ネットマスク等のインタフェースを設定するために使用されます。インタフェース上で複数のIPアドレスを設定するのに使用できます(マルチホーミング)。この関数は、インタフェースを使用する前に呼び出す必要があります。以下に例を示します。

```
errorCode = tfConfigInterface(myInterfaceHandle,
                              inet_addr ("208.229.201.65"),
                              inet_addr ("255.255.255.0"),
                              0,
                              1,
                              (unsigned char)0);
```

注意: インタフェースの最初の設定(multiHomeIndex = 0)にはtfConfigInterface関数ではなく、tfOpenInterface関数の使用を推奨します。

パラメータ	説明
interfaceHandle	インタフェースハンドル。これはtfKasagoInitializeにより返却される値です。
ipAddress	インタフェースの IP アドレス。
netMask	このデバイスのネットマスク。
flag	デバイスの特殊フラグ(以下参照)
buffersPerFrameCount	LANコントローラが分割送信機能をサポートしている場合、分割バッファの数を指定します。分割送信機能がサポートされていない場合は1を指定します。
multiHomeIndex	マルチホーミングで使用する IP アドレスのインデックス番号。最初のマルチホームインデックスは 0 でなければならない。

flag

値	意味
TM_DEV_SCATTER_SEND_ENB	このデバイスは、フレームごとに複数バッファによるデータ送信に対応する。このフラグを設定した場合は、buffersPerFrameCountを1より大きくする必要がある。
TM_DEV_MCAST_ENB	このデバイスはマルチキャストアドレスに対応する。
TM_DEV_IP_FORW_ENB	このデバイスのIP転送を許可する。
TM_DEV_IP_FORW_DBROAD_ENB	このデバイスのIPブロードキャストの転送を許可する。
TM_DEV_IP_FORW_MCAST_ENB	このデバイスのIPマルチキャストメッセージの転送を許可する。
TM_DEV_IP_BOOTP	BOOTPクライアントプロトコルを使ってIPアドレスを設定する。tfUseBootpは最初に呼び出す必要がある。
TM_DEV_IP_DHCP	DHCPクライアントプロトコルを使ってIPアドレスを設定する。tfUseDhcpは最初に呼び出す必要がある。

戻り値	意味
0	成功
TM_EADDRNOTAVAIL	ブロードキャストアドレス，クラスD ,Eなど不正なアドレスでデバイスを設定した。
TM_EINPROGRESS	tfConfigInterfaceコールは完了していない。このエラーは、DHCPやBOOTPの設定で返却される。
TM_ENOBUFS	オペレーションを完了できるだけのメモリがない。
TM_EINVAL	パラメータの1つが無効、または最初の設定がマルチホームインデックス0になっていない。ゼロのIPアドレスはBOOTP、またはDHCPフラグがオンの場合、およびPPPの場合にのみ許可される。それ以外はTM_EINVALが返却される。
TM_EALREADY	tfConfigInterfaceはまだ完了していない。
TM_EPERM	tfUseDhcp(またはtfUseBootp)を呼び出さずに、tfConfigInterfaceをコールした。(TM_DEV_IP_BOOTP、またはTM_DEV_IP_DHCP指定時のみ)
TM_ETIMEDOUT	DHCP、またはBOOTP要求がタイムアウトした。
TM_EAGAIN	PPPセッションは現在クローズ中である。
その他	デバイスドライバオープン関数で返却されたエラー値。

tfGetIpAddress

```
int                         tfGetIpAddress
(
    ttUserInterface     interfaceHandle,
    ttUserIpAddress * ifIpAddressPtr,
    unsigned char       mutiHomeIndex
);
```

関数の説明

この関数は、指定のインタフェースからIPアドレスを取得するために使用されます。マルチホームインデックスは、1つ以上のIPアドレスを持つインタフェースに使用されます。インタフェースがIPアドレスを1つしか持っていない場合は、マルチホームインデックスを0に設定しなければなりません。

パラメータ	説明
interfaceHandle	IPアドレスを取得するインタフェースハンドル
ifIpAddressPtr	インタフェースのIPアドレスを格納するバッファへのポインタ。
mutiHomeIndex	複数の IP アドレスに対するインデックス番号。

戻り値	意味
0	成功
TM_EINVAL	パラメータの1つが無効。
TM_ENETDOWN	インタフェース/マルチホームインデックスが設定されていない。

tfOpenInterface

```
#include <trsocket.h>

int                     tfOpenInterface
(
    ttUserInterface interfaceHandle,
    ttUserIpAddress ipAddress,
    ttUserIpAddress netMask,
    int             flag,
    int             buffersPerFrameCount
);
```

関数の説明

この関数は、IPアドレス、ネットマスク等のインタフェースを設定するために使用されます。この関数は、インタフェースを使用する前に呼び出す必要があります。以下に例を示します。

```
errorCode = tfOpenInterface(myInterfaceHandle,
                            inet_addr ("208.229.201.65"),
                            inet_addr ("255.255.255.0"),
                            0,
                            1);
```

注意： インタフェースの最初の設定にはtfConfigInterface関数ではなく、tfOpenInterface関数の使用を推奨します。

パラメータ	説明
interfaceIdHandle	インタフェースハンドル。これはtfKasagoInitializeにより返却される値です。
ipAddress	インタフェースのIPアドレス。
netMask	このデバイスのネットマスク。
flag	デバイスの特殊フラグ(以下参照)。
buffersPerFrameCount	LAN コントローラが分割送信機能をサポートしている場合、分割バッファの数を指定します。分割送信機能がサポートされていない場合は 1 を指定します。

flag

値	意味
TM_DEV_SCATTER_SEND_ENB	このデバイスは、フレームごとに複数バッファによるデータ送信に対応する。このフラグを設定した場合は、buffersPerFrameCountを1より大きくする必要がある。
TM_DEV_MCAST_ENB	このデバイスはマルチキャストアドレスに対応する。
TM_DEV_IP_FORW_ENB	このデバイスのIP転送を許可する。
TM_DEV_IP_FORW_DBROAD_ENB	このデバイスのIPブロードキャストの転送を許可する。
TM_DEV_IP_FORW_MCAST_ENB	このデバイスのIPマルチキャストメッセージの転送を許可する。
TM_DEV_IP_BOOTP	BOOTPクライアントプロトコルを使ってIPアドレスを設定する。tfUseBootpは最初に呼び出す必要がある。
TM_DEV_IP_DHCP	DHCPクライアントプロトコルを使ってIPアドレスを設定する。tfUseDhcpは最初に呼び出す必要がある。

戻り値	意味
0	成功
TM_EADDRNOTAVAIL	ブロードキャストアドレス，クラスD ,Eなど不正なアドレスでデバイスを設定した。
TM_EINPROGRESS	tfOpenInterfaceコールは完了していない。このエラーは、DHCPやBOOTPの設定で返却される。
TM_ENOBUFS	オペレーションを完了できるだけのメモリがない。
TM_EINVAL	パラメータの1つが無効。ゼロのIPアドレスはBOOTP、またはDHCPフラグがオンの場合、およびPPPの場合にのみ許可される。それ以外はTM_EINVALが返却される。
TM_EALREADY	tfOpenInterfaceはまだ完了していない。
TM_EPERM	tfUseDhcp (またはtfUseBootp)を呼び出さずに、tfOpenInterfaceをコールした。(TM_DEV_IP_BOOTP、またはTM_DEV_IP_DHCP指定時のみ)
TM_ETIMEDOUT	DHCP、またはBOOTP要求がタイムアウトになった。
TM_EAGAIN	PPPセッションは現在クローズ中である。
その他	デバイスドライバオープン関数で返却されたエラー値。

tfUnConfigInterface

```
#include <trsocket.h>

int                     tfUnConfigInterface
(
    ttUserInterface interfaceHandle,
    unsigned char   multiHomeIndex
);
```

関数の説明

この関数は、ある特定のマルチホームインデックスのインタフェースから、IPアドレスを削除するために使用されます。インタフェースを非設定にすることによって、新しいIPアドレスとネットマスクで再設定することが出来ます。

パラメータ	説明
interfaceHandle	デバイスのインタフェースハンドル。
multiHomeIndex	IPアドレスのインデックス番号。

戻り値	意味
0	成功
TM_EINVAL	パラメータの1つが無効。
TM_ENOENT	インタフェースは組み込まれていなかった。

ARP/ルーティングテーブル API

tfAddArpEntry

```
#include <trsocket.h>

int                     tfAddArpEntry
(
    ttUserIpAddress arpIpAddress,
    char *          physAddrPtr,
    int             physAddrLength
);
```

関数の説明

この関数は、ARPキャッシュへエントリを追加するために使用されます。この関数で設定されたエントリはTCP/IPスタックによって無限に保持されます。

パラメータ	説明
arpIpAddress	ARPキャッシュに追加するIPアドレス
physAddrPtr	物理アドレスが格納されているバッファへのポインタ。
physAddrLength	物理アドレスの長さ。

戻り値	意味
0	成功
TM_EHOSTUNREACH	宛先ホストへの経路がない。
TM_EINVAL	パラメータの 1 つが無効。
TM_ENOBUFS	オペレーションを完了するだけのメモリがない。

tfAddDefaultGateway

```
#include <trsocket.h>

int                 tfAddDefaultGateway
(
    ttUserInterface interfaceHandle,
    ttUserIpAddress gatewayIpAddress
);
```

関数の説明

この関数は、全てのインタフェースのデフォルトゲートウェイを設定するために使用されます。

パラメータ	説明
interfaceHandle	デフォルトゲートウェイを追加するインタフェースハンドル
gatewayIpAddress	ネットワークバイト順のデフォルトゲートウェイのIPアドレス。

戻り値	意味
0	成功
TM_EINVAL	パラメータの1つが無効。
TM_ENOBUFS	ルーティングエントリを割当てるだけのバッファがない。
TM_EALREADY	デフォルトゲートウェイが既にルーティングテーブルにある。
TM_EHOSTUNREACH	無効なゲートウェイアドレス。
TM_ENOENT	まだインタフェースはオープンされていない。 (R4.0.2.13以降でTM_SINGLE_INTERFACE_HOME未定義時)
TM_ENETDOWN	まだインタフェースはオープンされていない。 (R4.0.2.12以前，及びR4.0.2.13以降でTM_SINGLE_INTERFACE_HOME定義時)

tfAddStaticRoute

```
#include <trsocket.h>

int                     tfAddStaticRoute
(
    ttUserInterface interfaceHandle,
    ttUserIpAddress destIpAddress,
    ttUserIpAddress destNetMask,
    ttUserIpAddress gateway,
    int             hops
);
```

関数の説明

この関数は、インタフェースにルーティング情報を追加するために使用されます。

パラメータ	説明
interfaceHandle	ルーティング情報を追加するインタフェースハンドル。
destIpAddress	ルートを追加する宛先のIPアドレス。
destNetMask	ルートのネットマスク。
gateway	該当するルートに対応するゲートウェイのIPアドレス。
hops	ホストとルート間にあるルート数。

戻り値	意味
0	成功
TM_EINVAL	パラメータの1つが無効。
TM_EHOSTUNREACH	無効なゲートウェイアドレス。
TM_EALREADY	指定のアドレスがすでにルーティングテーブルにある。
TM_ENOBUFS	ルーティング情報を割当てるだけのバッファがない。
TM_ENETDOWN	まだインタフェースはオープンされていない。

PING アプリケーションプログラムインタフェース

PINGアプリケーションプログラムインタフェースには3つの関数が用意されています。

ユーザは最初に**tfPingOpenStart**をコールします。この関数はICMPソケットを開いて、定期的にPINGエコー要求パケットを送信します。この関数のパラメータには、リモートホストのIPアドレスを含む文字列へのポインタ(小数点で区切られた10進アドレス表記)、PINGエコー要求の間隔(秒単位)、PINGエコー要求のユーザデータの長さ、及びPINGエコー応答，またはICMPエラーメッセージが受信されたときに呼び出されるコールバック関数へのポインタを指定します。**tfPingOpenStart**は成功すると、ソケットディスクリプタを返却します。KASAGOスタックは、このソケットディスクリプタをパラメータとする**tfPingClose**が呼び出されるまで、タイマーによりPINGエコー要求を送信し続けます。

tfPingCloseを呼び出す前に、ユーザはPINGを行った結果、送信されたパケット数、受信されたパケット数、最後のPINGエコー要求の往復時間、受信したICMPエラーメッセージのエラーコードなどの統計情報を取得するために、**tfPingGetStatistics**を呼び出すことが出来ます。**tfPingGetStatistics**の第１のパラメータには**tfPingOpenStart**によって返却されるソケットディスクリプタ、第２のパラメータには、ttPingInfo構造体へのポインタを指定します。

ユーザが**tfPingOpenStart**の最終パラメータにコールバック関数を指定すると、PINGエコー応答またはICMPエラーメッセージの受信がユーザに通知されます。この場合、PINGエコー応答、またはICMPエラーメッセージを受信するたびにコールバック関数が呼び出されます。コールバック関数には、**tfPingOpenStart**で返却されたソケットディスクリプタが渡されます。この時ユーザは、**tfPingGetStatistics**を呼び出すことによりPINGの統計情報を取得することが出来ます。

tfPingClose

```
#include <trsocket.h>

int     tfPingClose
(
    int socketDescriptor
);
```

関数の説明

この関数は、PINGエコー要求の送信を中止して、tfPingOpenStartによって開かれているICMPソケットを閉じます。

注意： tfPingOpenStartで指定したコールバック関数から、この関数をコールすることはできません。

パラメータ	説明
socketDescriptor	tfPingOpenStartによって返却されたICMPのPINGソケットディスクリプタ。

戻り値	意味
0	成功
-1	エラーが発生

tfPingClose が失敗した場合、tfGetSocketError（socketDescriptor)で、対応するエラーコードを取得することが出来ます。

TM_EBADF	ソケットディスクリプタが無効

tfPingGetStatistics

```
#include <trsocket.h>

int                 tfPingGetStatistics
(
    int             socketDescriptor,
    ttPingInfoPtr pingInfoPtr
);
```

関数の説明

この関数は、PING統計情報(pingInfoPtrが示すttPingInfo構造体に格納されます)を取得します。
socketDescriptorにはtfPingOpenStartによって返却されたソケットディスクリプタを指定します。
pingInfoPtrには、ユーザが割り当てたttPingInfo構造体へのポインタを指定します。

パラメータ	説明
socketDescriptor	tfPingOpenStartによって返却されたICMPのPINGソケットディスクリプタ。
pingInfoPtr	呼出しが成功した場合に PING 統計情報が格納される構造体へのポインタ(以下参照)。

ttPingInfo データ構造体
（往復時間の最小単位は、タイマ更新間隔(TM_TICK_LENGTH)になります。）

メンバ	データタイプ	説明
pgiTransmitted	unsigned long	これまでに送信したPINGエコー要求のパケット数。
pgiReceived	unsigned long	これまでに受信したPINGエコー応答のパケット数。(重複は含まない)
pgiDuplicated	unsigned long	これまでに受信したPINGエコー応答のパケットのうち、重複パケットの数。
pgiLastRtt	unsigned long	最後のPING要求／応答の往復時間。ミリ秒単位。
pgiMaxRtt	unsigned long	PING要求/応答パケットの最大往復時間。ミリ秒単位。
pgiMinRtt	unsigned long	PING要求/応答パケットの最小往復時間。ミリ秒単位。
pgiAvgRtt	unsigned long	PING要求/応答パケットの平均往復時間。ミリ秒単位。
pgiSumRtt	unsigned long	PING要求/応答パケットの合計往復時間。ミリ秒単位。
pgiSendErrorCode	int	PING送信要求エラーコード。
pgiRecvErrorCode	int	PING受信エラーコード。

戻り値	意味
0	成功
-1	エラーが発生

tfPingGetStatisticsが失敗した場合、tfGetSocketError(socketDescriptor)で、対応するエラーコードを取得することができます。

TM_EBADF	socketDescriptorが無効。
TM_EINVAL	pingInfoPtrがヌルポインタ。
TM_EINVAL	socketDescriptor は tfPingOpenStart で開かれていなかった。

tfPingOpenStart

```
#include <trsocket.h>

int                     tfPingOpenStart
(
    char *              remoteHostNamePtr,
    int                 pingInterval,
    int                 pingDataLength,
    ttPingCBFuncPtr pingUserCBFuncPtr
);
```

関数の説明

この関数はICMPソケットを開いて、remoteHostNamePtrで指定されたリモートホストへPINGエコー要求を送信し始めます。PINGエコー要求はpingIntervalで指定された秒間隔で送信されます。PINGの長さ(IP及び、ICMPヘッダは含まない)はpingDataLengthに指定します。

pingUserCBFuncPtrにコールバック関数を指定した場合は、PINGエコー応答とICMPエラーメッセージを受信する毎に呼び出されます。PINGの統計情報を取得するには、tfPingGetStatisticsを呼び出す必要があります。PINGエコー要求の送信を終了して、ICMPソケットを閉じるには、tfPingCloseを呼び出す必要があります。

パラメータ	説明
remoteHostNamePtr	小数点で区切られた10進IPアドレスを含む文字列へのポインタ。
pingInterval	PINGエコー要求の間隔(秒単位)。0の場合、デフォルトの1秒が設定される。
pingDataLength	PINGエコー要求のユーザデータの長さ。0の場合、デフォルトの56バイトが設定される。1から3の間の場合、4バイトに設定される。
pingUserCBFuncPtr	PINGエコー応答、またはICMPエラーメッセージが受信された時に、呼び出されるコールバック関数へのポインタ。ユーザが受信の通知を望まない場合、ヌル関数ポインタに設定することができる。

戻り値

新しいICMPソケットディスクリプタ、失敗した場合は、TM_SOCKET_ERROR (-1)が返却されます。

tfPingOpenStartが失敗した場合、tfGetSocketError(TM_SOCKET_ERROR)でエラーコードを取得することが出来ます。

TM_EINVAL	remoteHostNamePtrがヌルポインタ
TM_EINVAL	pingIntervalが無効
TM_EINVAL	pingDataLengthがIPプロトコルが使用できる最大値65507よりも大きい。
TM_ENOBUFS	オペレーションを完了するだけのメモリがない。
TM_EMFILE	使用可能なソケットがこれ以上ない。
TM_EMSGSIZE	pingDataLengthがソケット送信キューの制限値を超えている。またはpingDataLengthがIP MTUを超えていて、フラグメントが認められていない。
TM_EHOSTUNREACH	リモートホストへのルートがない。

ヌル以外の関数ポインタの使用例

```
#include <trsocket.h>

void tfPingUserCB (int socketDescriptor);
..
socketDescriptor = tfPingOpenStart (“192.168.0.2”, 0, 0, tfPingUserCB);
```

ヌル関数ポインタの使用例

```
#include <trsocket.h>
..
socketDescriptor = tfPingOpenStart (“127.0.0.1”, 0, 0, ttPingCBFuncPtr)0);
.
```

組込みシステム開発専用　TCP/IP　プロトコルスタック
KASAGO TCP/IP
ユーザーズマニュアル

無償版　2014年01月28日

図研エルミック株式会社

本社　〒222-8505　横浜市港北区新横浜 3 丁目 1 番 1 号
図研新横浜ビル 2 階
TEL: 045-624-8111（代）
FAX: 045-470-3018

大阪営業所　〒556-0017　大阪市浪速区湊町 1 丁目 4 番 38 号
近鉄新難波ビル 9 階
TEL: 06-4396-8430（代）
FAX: 06-4396-8431

URL: http://www.elwsc.co.jp