RX ファミリ

組み込み用 TCP/IP M3S-T4-Tiny ソケット API モジュール

Firmware Integration Technology

R20AN0296JJ0131
Rev.1.31
2016.10.01

要旨

このソフトウェアは組み込み用 TCP/IP M3S-T4-Tiny(以下 T4)用のソケット API モジュールです。T4 は ITRON TCP/IP API に対応しています。一方、多くの地域で幅広く使われているネットワーク用 API はソケット API です。より多くのユーザが T4 用アプリケーションを開発できるように、T4 用の簡易ソケット API を用意しました。ユーザは T4 に加えて本モジュールを使用することでソケット API を使用することが出来ます。

T4 についての情報は以下 URL をご参照ください。

https://www.renesas.com/mw/t4

ソケット API と T4 は FIT モジュールとして提供されます。FIT モジュールについては以下 URL をご参照ください。

https://www.renesas.com/ja-jp/solutions/rx-applications/fit.html

以下の図は T4 を使用したソフトウェア構造、2 種類の例です。

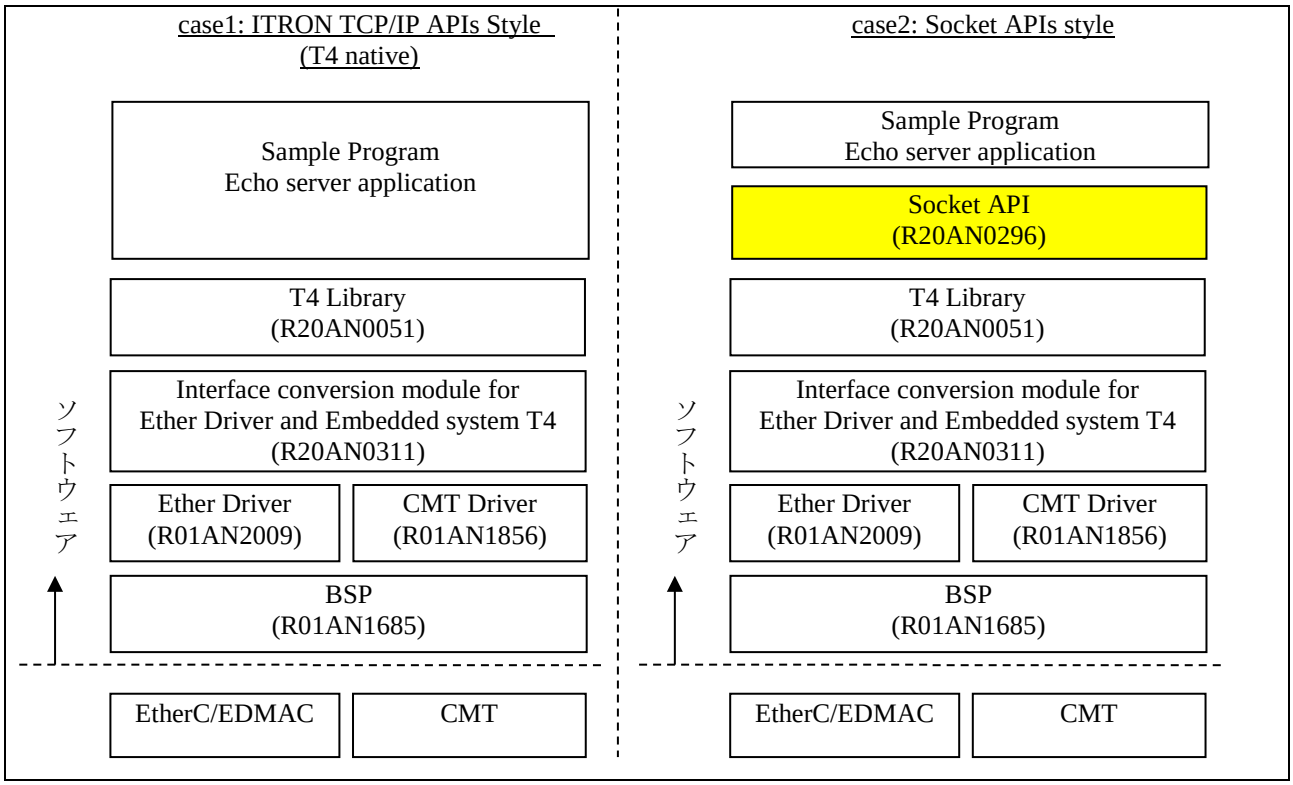

図 1　T4 ソフトウェア構成

注意事項：

本ソケット API は簡易実装のため、ソケット API の基本機能のみ提供します。Apache 等のソケット API を使用したアプリケーションをそのまま移植することは出来ません。

動作確認デバイス

RX ファミリ

目次

1. 概要

1.1 ソケット API と T4 API の対応表

以下にソケット API と T4 API の対応表を示します。

表 1 ソケット API と T4 API の対応表

No	機能説明	ソケット API	対応する T4 API
1	ソケット API を開始します。	R_SOCKET_Open()	tcpudp_get_ramsize() tcpudp_open()
2	ソケット API を終了します。	R_SOCKET_Close()	tcpudp_close()
3	ソケットを生成し正数値の ID を割り当て、システムリソースを確保します。	socket()	get_random_number()
4	生成したソケットに対し accept()で待受けるポート番号を設定することが出来ます。	bind()	-
5	TCP クライアントが使用する関数です。フリーのローカルポートを使用し通信相手と接続します。 UDP として使用する場合、通信相手の IP アドレスとポート番号を固定化します。	connect()	tcp_con_cep() get_random_number()
6	TCP サーバが使用する関数です。LISTEN 状態に遷移します。	listen()	tcp_acp_cep()
7	TCP サーバが使用する関数です。新しい TCP 接続を受け付けることが出来ます。TCP 接続は通信相手である TCP クライアントから来ます。	accept()	tcp_acp_cep() tcp_rcv_dat()
8	送信データをソケットに書き込みます。	send()	tcp_can_cep() tcp_snd_dat()
9	送信データを宛先情報と共にソケットに書き込みます。ソケットは SOCK_DGRAM(UDP)で生成されている必要が有ります。	sendto()	udp_snd_dat()
10	ソケットから受信データを読み込みます。	recv()	tcp_rcv_dat()
11	ソケットから受信データと宛先情報を読み込みます。ソケットは SOCK_DGRAM(UDP)で生成されている必要が有ります。	recvfrom()	
12	送信を終了します。	-	tcp_sht_cep()
13	ソケットを閉じます。	closesocket()	tcp_can_cep() tcp_cls_cep() udp_can_cep()
14	ソケットの設定を変更します。	fcntl()	-
15	複数ソケットの状態を調べます。	select()	tcpudp_get_time()

2. API 情報

本モジュールの API はルネサスの API の命名基準に従っています。

2.1 ハードウェアの要求

なし

2.2 ソフトウェアの要求

このドライバは以下のパッケージに依存しています。

- r_bsp
- r_t4_rx
- r_t4_driver_rx

2.3 サポートされているツールチェイン

このドライバは下記ツールチェインで動作確認を行っています。

- Renesas RX Toolchain v.2.05.00

2.4 ヘッダファイル

すべての API 呼び出しと使用されるインタフェース定義は r_socket_rx_if.h に記載しています。

2.5 整数型

より分かりやすい、移植性の高いコードのため、このプロジェクトは ANSI C99「正確な幅の整数型」を使用しています。これらの型は stdint.h で定義されています。

2.6 コンフィグレーション

本モジュールのコンフィギュレーションオプションの設定は、r_socket_rx_config.h で行います。

オプション名および設定値に関する説明を、下表に示します。

表 2 コンフィギュレーションオプション

コンフィグレーション内容 *r_socket_rx_config.h*	
#define MAX_UDP_CCEP - Default value = 4	T4 によって確保される UDP 通信端点(エンドポイント)の個数。T4 のコンフィグファイル”config_tcpudp.c”の udp_ccep 構造体のメンバ数に合わせた値で設定してください。
#define MAX_TCP_CCEP - Default value = 4	T4 によって確保される TCP 通信端点(エンドポイント)の個数。T4 のコンフィグファイル"config_tcpudp.c"の tcp_ccep 構造体のメンバ数に合わせた値で設定してください。 MAX_TCP_CCEP は 2 以上を設定してください。
#define MAX_TCP_CREP - Default value = MAX_TCP_CCEP	T4 によって確保される TCP 受付口の個数。標準では TCP 通信端点の個数と同じ個数を割り当てています。
#define SOCKET_TCP_WINSIZE - Default value = 1460	T4 が使用する TCP ウインドウサイズ。
#define TCPUDP_WORK - Default value = 7200	T4 が使用するワーク領域のサイズ。このワーク領域のサイズはソケットの個数に依存します。このワーク領域の必要サイズは T4 の API "tcpudp_get_ramsize()"により調べることが出来ます。デフォルト値の 7200 バイトは、MAX_TCP_CCEP=4, MAX_UDP_CCEP=4 を設定した時の値です。
#define TOTAL_BSD_SOCKET - Default value = (MAX_UDP_CCEP+ MAX_TCP_CCEP)	ソケットの合計数です。このパラメータは T4 の通信端点構造体(tcp_ccep[]と udp_ccep[])のメンバ数の合計値です。
#define SOCKET_IF_USE_SEMP - Default value = 0	ロック機構かセマフォの機構が搭載されている場合、1 に設定してください。これは socket() API を同時呼び出しした場合の動作を保証します。
#define R_SOCKET_PAR_CHECK - Default value = 1	ソケット API のパラメータチェックを省略したい場合、この define 定義を #undef で無効化してください。
#define BSD_RCV_BUFSZ - Default value = 1460	ソケットで受信したデータを格納するために使用される受信バッファのサイズ。
#define BSD_SND_BUFSZ - Default value = 1460	ソケットで送信されるデータを格納するのに使用される送信バッファのサイズ。

2.7 API データ構造

API 関数の引数である構造体を示します。

```
struct sockaddr {
    unsigned short sa_family;    /* address family, AF_xxx */
    char          sa_data[14];  /* up to 14 bytes of direct address */
};

struct in_addr {
    union
    {
        struct
        {
            unsigned char s_b1,s_b2,s_b3,s_b4;
        } S_un_b;
        struct
        {
            unsigned short s_w1,s_w2;
        } S_un_w;
        unsigned long S_addr;
    } S_un;
};

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

```
typedef struct _tagfd_set {
    __fd_mask fds_bits[__howmany(FD_SIZE, __NFDBITS)];
} fd_set;
```

```
struct timeval
{
    long tv_sec;
    long tv_usec;
};
```

2.8 戻り値

API 関数の戻り値を示します。 これらは全て、r_socket_rx_if.h で定義されています。

```
/**** Return values for functions ****/
/* Socket does not exist */
#define INVALID_SOCK (-1)
#define INVALID_SOCKET (-1)
/* Operation failed */
#define SOCKET_ERROR (-1)
/* No memory is available to allocate packet buffer */
#define SOCKET_BFR_ALLOC_ERROR (-2)
/* No connection between network and the host */
#define SOCKET_HOST_NO_ROUTE (-3)
/* Socket transmission length exceed size of data buffer */
#define SOCKET_MAX_LEN_ERROR (-4)
/* Socket is not ready for transmission */
#define SOCKET_NOT_READY (-5)
/* Socket is not ready for transmission. For backward compatibility */
#define SOCKET_TX_NOT_READY (-5)
/* Socket connection has not yet been established */
#define SOCKET_CNXN_IN_PROGRESS (-6)
/* Parameter error */
#define E_PAR (-33)
```

2.9 エラーコード

以下に、ソケット API で使用される全てのエラーコードを示します。

表 3　エラーコード

エラーコード	値	説明
ENFILE	23	利用可能なファイル記述子がありません。
EAGAIN	35	非ブロック要求が受け入れられました。
EINPROGRESS	36	即時に接続することができません。
EALREADY	37	要求されたソケットは使用中です。
ENOTSOCK	38	無効なソケットです。
EDESTADDRREQ	39	ソケットがローカルアドレスにバインドできません。
EPROTOTYPE	41	ソケットタイプがサポートされていません。
EPROTONOSUPPORT	43	プロトコルがサポートされていません。
EOPNOTSUPP	45	ソケットはリスニングモードの状態で、接続することができません。
EAFNOSUPPORT	47	アドレスファミリがサポートされていません。
ECONNABORTED	53	接続が中止されました
ECONNRESET	54	接続が接続先によって強制的にクローズされました。
EISCONN	56	指定されたソケットは既に接続されています。
ENOTCONN	57	指定されたソケットが接続されていません。
ETIMEDOUT	60	操作がタイムアウトしました。
ENODATA	61	送信するデータがありません。

2.10 モジュールの追加方法

本モジュールは既存の e² studio プロジェクトに追加する必要があります。e² studio plug-in を使用することによって自動的にインクルードファイルパスを更新することができるため、プロジェクトへの追加には plug-in の使用を推奨します。他の方法として、本アプリケーションノートに付随するアーカイブからモジュールを手動でインポートすることも可能です。手動での追加手順は下記の通りです。

1. 本アプリケーションノートと共に、r_socket_rx フォルダ内にモジュール本体を含む「組み込み用 TCP/IP M3S-T4-Tiny ソケット API モジュール」の ZIP ファイルパッケージが配布されています。
2. 任意のフォルダにパッケージを解凍してください。
3. ファイルブラウザ上で、ZIP ファイルを解凍したフォルダを開き、r_socket_rx フォルダを見つけてください。
4. e²studio ワークスペースを開いてください。
5. e²studio のプロジェクトエクスプローラウィンドウでソケットモジュールを追加したいプロジェクトを選択してください。
6. r_socket_rx フォルダをファイルブラウザから e2studio プロジェクトの最上位にドラッグ＆ドロップ（又はコピー・貼り付 け）してください。
7. プロジェクトのインクルードパスに、モジュールファイルへのパスを追加してください。：
   a. 「ディレクトリのパスの追加」コントロールに移動してください。
      i. 'project name'->properties->C/C++ Build->Settings->Compiler->Source -Add (green +icon)
      (プロジェクト名->プロパティ->C/ C++ビルド->設定->コンパイル>ソース-Add(緑+アイコン)
   b. 下記のパスを追加してください。
      i. "${workspace_loc:/${ProjName}/r_socket_rx}"
      ii. "${workspace_loc:/${ProjName}/r_socket_rx/src}"
      プラグインを使用したか、または手動でプロジェクトにパッケージを追加したかにかかわらず、アプリケーション用にモジュールのコンフィギュレーションが必要です。
8. プロジェクト内で,移動元フォルダーr_socket_rx/ref/から r_socket_rx_config_reference.h ファイルを探し,プロジェクトの r_config フォルダへコピーします。
9. コピーされた r_config フォルダ内のファイルを r_socket_rx_config.h にリネームします。
10. コピーされた r_socket_rx_config.h ファイルを編集することによって,必要なコンフィギュレーションを行ってください。第 2.6 章のコンフィギュレーションを参照してください。

3. API 関数

3.1 概要

表 4 ソケット API Function 一覧

Function	Description
R_SOCKET_Open()	すべてのソケット構造体を初期化します。
R_SOCKET_Close()	ソケット API の動作を終了します。
socket()	新しいソケットを作成します。
bind()	ローカルアドレスにソケットをバインドします。
connect()	サーバ側に接続を要求します。
listen()	ソケットを LISTEN 状態に遷移します。
accept()	クライアント側から接続を受け入れます。
send()	データをストリームソケットに送信します。
sendto()	データをデータグラムソケットに送信します。
recv()	ストリームソケットからデータを受信します。
recvfrom()	データグラムケットからデータを受信します。
closesocket()	ソケットを閉じます。
fcntl()	ソケットのタイムアウト値を変更します。
select()	I/O 多重化を同期させます。

3.2 R_SOCKET_Open()

ソケット構造体を初期化します。

Format

```
void R_SOCKET_Open( void )
```

Parameters

None.

Return Values

None.

Properties

Prototyped in r_socket_rx_if.h.

Description

ソケット構造体を初期化します。T4 通信端点、CCEP 構造体の rbufsz も合わせて初期化した後、tcpudp_open()を呼び出します。tcpudp_open()では R_SOCKET_Open()で指定された rbufsz の値を使用し tcupdp_work から受信バッファを割り当てます。

Reentrant

なし

Examples

```
R_SOCKET_Open();
```

Special Notes:

この API は tcp_ccep[]構造体を初期化し、この構造体は T4 で使用される TCP 通信端点(エンドポイント)として使用されます。T4 の tcpudp_open() は、この構造体の設定値を用いて T4 のワーク領域(tcpudp_work[])からバッファ領域を割り当てます。最後に tcpdudp_open() を呼び出します。また、ネットワーク層の初期化関数である lan_open()も合わせて呼び出してください。lan_open()、R_SOCKET_Open()の順序で呼び出してください。

3.3 R_SOCKET_Close()

ソケット API の動作を終了します。

Format

```
void R_SOCKET_Close( void )
```

Parameters

None.

Return Values

None.

Properties

Prototyped in r_socket_rx_if.h.

Description

ソケット API の動作を終了します。ユーザは本関数を呼ぶ前に生成したすべてのソケットに対して closesocket()を呼び出してクローズしてください。

Reentrant

なし

Examples

```
R_SOCKET_Close();
```

Special Notes:

3.4 socket()

新しいソケットを生成します。

Format

```
int socket( int domain, int type, int protocol )
```

Parameters

domain

AF_INET が受け付け可能です。他の値を指定すると SOCKET_ERROR が戻ります。

type

SOCK_STREAM を指定すると TCP ソケットとしてソケットを生成します。

SOCK_DGRAM を指定すると UDP ソケットとしてソケットを生成します。

protocol

type が SOCK_DGRAM であれば、IPPROTO_UDP をセットしてください。

または、type が SOCK_STREAM であれば、IPPROTO_TCP を設定してください。

Return Values

SOCKET_ERROR	*処理失敗; エラータイプを示す errno をチェックしてください*
E_PAR	*パラメータエラー*
0 or Positive value	*処理が成功し、ソケット ID が戻ります。*

Error Types

EAFNOSUPPORT	*アド レスファミリがサポートされていません。*
EPROTOTYPE	*ソケットタイプはサポートされていません。*
ENFILE	*利用可能なソケットがありません。*
EPROTONOSUPPORT	*プロトコルがサポートされていません。*

Properties

Prototyped in r_socket_rx_if.h.

Description

新しいソケットを生成します。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Example

```
int32_t  sock1, err;

sock1 = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP);
if( sock1 == SOCKET_ERROR )
{
   /*… error handling …*/
}
```

Special Notes:

ソケット番号{0 … MAX_TCP_CCEP-1} は、TCP によって使用されます。ソケット番号 {MAX_TCP_CCEP…(MAX_TCP_CCEP+MAX_UDP_CCEP-1)}は、UDP によって使用されます。

socket()の同時呼び出しに対する問題を回避する必要が有る場合、ロック機構を実装してください。

3.5 bind()

生成したポートに accept()で待受けるポート番号を設定することが出来ます。

Format

```
int bind( int sock, const struct sockaddr * name, int namelen )
```

Parameters

sock

ソケット ID

name

sockaddr 構造体へのポインタです。構造体にはローカルアドレス情報を格納してください。

namelen

sockaddr 構造体のデータ長を格納してください。

Return Values

SOCKET_ERROR	*処理失敗; エラータイプを示す errno をチェックしてください。*
E_PAR	*パラメータエラー*
E_OK	*処理成功*

Error Types

ENOTSOCK	*sock 引数はソケットを参照していません。*
EADDRNOTAVAIL	*指定されたローカルアドレスは利用可能ではありません。*
EINVAL	*ソケットが既にバインドされているか、プロトコルがバインドを必要としないか、またはソケットはシャットダウンされています。*
EPROTONOSUPPORT	*プロトコルはアドレスファミリーもしくは実装でサポートされません。*

Properties

Prototyped in r_socket_rx_if.h.

Description

生成したポートに accept()で待受けるポート番号を設定することが出来ます。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Example

```
SOCKET            sck;
struct sockaddr_in  serveraddr;

sck = socket(AF_INET, SOCK_STREAN, IPPROTO_TCP);
/* this is an Internet address */
serveraddr.sin_family = AF_INET;

/* let the system figure out our IP address */
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);

/* this is the port we will listen on */
serveraddr.sin_port = (unsigned short)(1234);

/*
 * bind: associate the socket, sck, with a port
 */
if (bind(sck, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) < 0)
{
   closesocket(sck);
   return SOCKET_ERROR;
}
```

3.6 connect()

通信相手に接続を開始します。

Format

```
int connect( int sock, struct sockaddr * name, int namelen )
```

Parameters

sock

ソケット ID

name

sockaddr 構造体へのポインタです。構造体には通信相手のアドレス情報(IP アドレスとポート番号)を格納してください。

namelen

sockaddr 構造体のデータ長を格納してください。

Return Values

SOCKET_ERROR	*処理失敗, エラータイプを示す errno をチェックしてください。*
E_PAR	*パラメータエラー*
E_OK	*処理成功*

Error Types

ENOTSOCK	*sock 引数はソケットを参照していません。*
EADDRNOTAVAIL	*指定されたローカルアドレスは利用可能ではありません。*
EALREADY	*指定されたソケットの接続要求は既に進行中です。*
EISCONN	*指定されたソケットは、接続モードであり、既に接続されています。*
EOPNOTSUPP	*ソケットは、正しい状態(LISTEN 中など)になく、接続できません*
EINVAL	*アドレス長がアドレスファミリに対して無効であるか、sockadr 構造体のアドレスファミリが無効です。*
EINPROGRESS	*O_NONBLOCK が、タイムアウトに設定されています。* *要求は非同期に実行されています。*
ETIMEDOUT	*接続が確立される前に、接続要求はタイムアウトしました。*
EPROTONOSUPPORT	*プロトコルはアドレスファミリもしくは実装でサポートされません。*

Properties

Prototyped in r_socket_rx_if.h.

Description

通信相手に接続を開始します。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Example

```
SOCKET           sck;
struct sockaddr_in  serveraddr;

sck = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
/* this is an Internet address */
serveraddr.sin_family = AF_INET;

/* let the system figure out our IP address */
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);

/* this is the port we will listen on */
serveraddr.sin_port = (unsigned short)(0);

/*
 * bind: associate the socket, sck, with a port
 */
if (bind(sck, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) < 0)
{
   closesocket(sck);
   return SOCKET_ERROR;
}
serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = 0xc0a80008; // 192.168.0.8
serveraddr.sin_port = (unsigned short)1024;

ercd = connect(sck, (struct sockaddr*)&serveraddr, sizeof(serveraddr));
```

Special Notes

ソケット非ブロッキングモードでは、TMO_NBLK は BSD ソケットの構造の TMOUT 引数に設定されています。connect() API が呼び出されると、接続がすぐに確立できない場合、connect() API が SOCKET_ERROR を返し、EINPROGRESS に errno を設定します。接続要求が中止されることはありませんが、接続が非同期的に確立されます。接続が確立される前に、後続の呼び出しは、同じソケットの接続することに失敗し、EALREADY を errno に設定します。

3.7 listen()

LISTEN 状態に遷移します。

Format

```
int listen( int sock, int backlog )
```

Parameters

sock

ソケット ID

backlog

1 をセットしてください。

本来の使用方法：キューイングする接続数の最大値を指定してください。(未実装)

Return Values

SOCKET_ERROR	*処理失敗; エラータイプを示す errno をチェックしてください*
E_PAR	*パラメータエラー*
E_OK	*処理成功*

Error Types

ENOTSOCK	*sock 引数はソケットを参照していません。*
ENOBUFS	*システムで利用可能なリソースが不足しています。*
EINVAL	*ソケットはシャットダウンされています。*
EDESTADDRREQ	*ソケットはローカルアドレスにバインドされておらず、プロトコルはバインドされていないソケットに対する LISTEN をサポートしていません。*
EOPNOTSUPP	*ソケットは、正しい状態(LISTEN 中など)になく、接続できません。*

Properties

Prototyped in r_socket_rx_if.h.

Description

listen 関数は指定されたソケットを LISTEN 状態に設定します。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Example

```
/*… After binding …*/

/*
 * listen: make this socket ready to accept connection requests
 */
if (listen(sck, 1) < 0) /* allow 1 requests to queue up */
{
   closesocket(sck);
   return SOCKET_ERROR;
}
```

Special Notes

ノンブロッキングモードでは、もう一つのソケットが内部的に確保され BSD_CONNECTING 状態のソケットになります。このソケットは接続を待受けます。ソケットの予備が無い場合、SOCKET_ERROR が戻り、 errno = ENFILE となります。

3.8 accept()

LISTEN 状態にあるソケットを接続要求受付可能な状態にします。

Format

```
int accept( int sock, struct sockaddr * address, int * address_len )
```

Parameters

sock

ソケット ID

address

sockaddr 構造体へのポインタです。構造体には通信相手のアドレス情報(IP アドレスとポート番号)が格納されます。ユーザが値を格納する必要はありません。

address_len

入出力両用の引数であり、呼び出し時には address_len によって参照されるバッファサイズが設定されている必要があります。 関数呼び出し後は、address_len に実際のデータサイズが格納されます。

Return Values

SOCKET_ERROR	*処理失敗 、エラータイプを示す errno をチェックしてください。*
E_PAR	*パラメータエラー*
Positive Value	*処理成功, 接続完了したソケット ID が返ります。*

Error Types

ECONNABORTED	*接続は中止されました。*
ENOTSOCK	*sock 引数はソケットを参照していません。*
EADDRNOTAVAIL	*指定されたローカルアドレスは利用可能ではありません。*
EAGAIN	*ソケットファイル記述子に O_NONBLOCK が設定されており、かつ、受け入れるべき接続が存在しません。*
EINVAL	*ソケットは接続を受け入れていません。*
EOPNOTSUPP	*指定されたソケットのソケットタイプは、accept による接続をサポートしていません。*

Properties

Prototyped in r_socket_rx_if.h.

Description

accept 関数は、LISTEN 状態のソケットからキューに入っている接続要求を accept するために使用されます。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Example

```
SOCKET            parent_sock, child_sock;
struct sockaddr_in  serveraddr;
struct sockaddr    clientaddr;
int              clientlen;

/* after binding */
child_sock = accept(parent_sock, &clientaddr, &clientlen);
```

Special Notes

ノンブロッキングモードで、accept() API が呼ばれた時に accept されるべき接続がない場合は、accept() API は errno に EAGAIN を設定して即時に SOCKET_ERROR を返して終了します。 後から、接続が確立されたかどうか確かめるために select() API を呼び出す必要があります。

accept() API の戻り値が引数のソケットと同じ値である場合、そのソケットはそれ以上の接続を受け入れることができません。この状況を避けるために、期待される接続数よりも２個以上多いソケットを用意してください。例えば、期待される接続数が４の場合、６個のソケットを準備してください。１つは listen 用、もう１つは接続の待機用、残りの４つが accept() API による接続用です。

表 5　4 個のソケットによる accept (ノンブロッキング)

ソケットの役割	リスナー	待機	子ソケット	備考
ソケット状態	BSD_LISTENING	BSD_CONNECTING	BSD_CONNECTED	
Socket(), bind()	0	---	---	ソケット#0 が作成され、ローカルアドレスとポートにバインドされます。
listen()	0	1[1]	---	ソケット#0 は LISTEN モードになり、ソケット#1 が接続待ち状態になります。
初回 accept 後	0	2[1]	1	ソケット#1 が子ソケットとして戻り、ソケット#2 が接続待ち状態になります。
2 回目の accept 後	0	3[1]	2	ソケット#2 が子ソケットとして戻り、ソケット#3 が接続待ち状態になります。
3 回目の accept 後	0	-1[2]	3	ソケット#3 が子ソケットとして戻ります。空きソケットが無いため、接続待ちソケットは -1 となります。
4 回目の accept 時	0	-1[2]	SOCKET_ERROR errno = ENFILE	4 回目の accept で、SOCKET_ERROR が戻り、errno = ENFILE が設定されます。
処理の例： いくつかの処理の後、ソケット#2 がクローズされたとします。このとき、select() を呼び出すと、未使用のソケットを検出し、listen しているソケットに対して「読み込み可」フラグをセットします。これは、アプリケーションが accept() を実行可能であることを意味します。				
5 回目の accept 後	0	2[3]	SOCKET_ERROR errno = ENFILE	今度は、ソケット#2 が接続待ちソケットになります。accept() API は引き続き SOCKET_ERROR を返します。その後の accept()は、接続待ちソケット(例えばソケット#2)を返します。
リスナーソケット #0 をクローズ	---	---	---	ソケット#0 がクローズされると、接続待ちソケット(ここでは#2)もクローズされます。

[1] 次に使用可能なソケット。1,2,3 の順を想定しています。

[2] 全てのソケット(0,1,2,3)が使用中です。-1 は無効なソケット番号を示します。

[3] ソケット#2 はクローズされています。この時点では、accept() API によって接続待ちソケットとして利用可能です。

3.9 send()

接続状態のソケットに対してデータを送信します。(TCP)

Format

```
int send( int sock, const char * buffer, size_t length, int flags )
```

Parameters

sock

ソケット ID

buffer

送信データを含むアプリケーションデータバッファ

length

送信データサイズをバイト数で指定してください。最大値は 0x7fff です。

flags

メッセージフラグです。未実装の引数です。0 を指定してください。

Return Values

SOCKET_ERROR	*処理失敗; エラータイプを示す errno をチェックしてください*
E_PAR	*パラメータエラー*
Positive Value	*処理成功、送信データ長が戻ります。*

Error Types

ENOTCONN	*ソケットは接続されていません。*
ENOTSOCK	*sock 引数はソケットを参照していません。*
EADDRNOTAVAIL	*指定されたローカルアドレスは利用可能ではありません。*
ECONNRESET	*接続先から強制的に接続をクローズされました。*
EOPNOTSUPP	*socket 引数はフラグでセットされた値の 1 つまたは複数をサポートしないソケットに関連付けられています。*
ENOBUFS	*システムで利用可能なリソースが不足しています。*
EAGAIN	*ソケットファイル記述子に O_NONBLOCK が設定されており、かつ、待ち時間なしに指定されたデータの送信を完了できません。*
E_QOVR	*2 つ以上の要求が同時に同じソケット記述子で発行されています。*

Properties

Prototyped in r_socket_rx_if.h.

Description

接続状態のソケットに対してデータを送信します。

指定される値は SOCK_STREAM である必要があります。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Example 1: send() API operation in blocking mode

```
/* Socket operation in blocking mode */

int32_t  sock1, remain_len, send_len;
int8_t   buffer[1000], *pbuf;

/*… sock1 was created and TCP sessions established … */
pbuf = &buffer[0];
remain_len = 1000;
send_len = send( sock1, pbuf, remain_len, 0 );
```

Example 2: send() API operation in non-blocking mode

```
/* Socket operation in non-blocking mode */

int32_t  sock1, remain_len, send_len;
int8_t   buffer[1000], *pbuf;

/*… sock1 was set to non-blocking mode (O_NONBLOCK) */
/*… sock1 was created and TCP sessions established … */
pbuf = &buffer[0];
remain_len = 1000;

/* Call send() API */
send_len = send(sock1, pbuf, remain_len, 0);
if (remain_len == send_len)
{
   /* All data in buffer are copied to socket's transmit internal buffer */
   /* send() in non-blocking mode is accepted! */
   remain_len = 0; // Clear remain_len
}
else
{
   /* Handle error process */
}
```

Special Notes:

ノンブロッキングモードでは、send()は、ソケットの送信バッファに転送したバイト数を戻り値として戻します。この時点では実際にデータは送信されていません。もし送信バッファサイズ(BSD_SND_BUFSZ)より大きなサイズのデータ長を指定した場合、SOCKET_ERROR が戻り、errno = ENOBUFS となります。select()を使用してデータが送信されたことと、新しい送信が可能になったことを確認してください。

3.10 sendto()

データグラムタイプのソケット(UDP)に対しデータ送信を行います。

Format

```
   int sendto( int sock, const char * buffer, size_t length, int flags, const
struct sockaddr * to, int tolen )
```

Parameters

sock

ソケット ID

buffer

送信データを含むアプリケーションデータバッファ

length

送信データサイズをバイト数で指定してください。最大値は 0x7fff です。

flags

メッセージフラグです。未実装の引数です。0 を指定してください。

to

sockaddr 構造体へのポインタです。構造体には通信相手のアドレス情報(IP アドレスとポート番号)を格納してください。

tolen

sockaddr 構造体のデータ長を格納してください。

Return Values

SOCKET_ERROR	*処理失敗; エラータイプを示す errno をチェックしてください。*
E_PAR	*パラメータエラー*
Positive Value	*処理成功、送信したデータ長が戻ります。*

Error Types

EOPNOTSUPP	*socket 引数はフラグでセットされた値の 1 つまたは複数をサポートしないソケットに関連付けられています。*
ENOTCONN	*ソケットは接続されていません。*
ENOTSOCK	*sock 引数はソケットを参照していません。*
EADDRNOTAVAIL	*指定されたローカルアドレスは利用可能ではありません。*
ENOBUFS	*システムで利用可能なリソースが不足しています。*
ECONNRESET	*接続先から強制的に接続をクローズされました。*
EINVAL	*tolen 引数はアドレスファミリに対して、有効な長さではありません。*
EAGAIN	*ソケットファイル記述子に O_NONBLOCK が設定されており、かつ、待ち時間なしに指定されたデータの送信を完了できません。*

Properties

Prototyped in r_socket_rx_if.h.

Description

データグラムタイプのソケット(UDP)に対しデータ送信を行います。

指定される値は SOCK_DGRAM である必要があります。

呼び出し時には、受信側のアドレスとポート番号を指定する必要があります。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Example 1: sendto() API operation in blocking mode

```
/* Socket operation in blocking mode */
int32_t  sock1, remain_len, send_len;
int8_t   buffer[1000], *pbuf;
struct sockaddr dest;
int32_t  addr_len;

/*… sock1 was created and TCP sessions established … */
pbuf = &buffer[0];
remain_len = 1000;
/* set the destination addr and len */
while( remain_len > 0 ) {        // repeat sending until all data is sent
   send_len = sendto( sock1, pbuf, remain_len, 0, &dest, addr_len );
   pbuf += send_len;
   remain_len -= send_len;
}
```

Example2: sendto() API operation in non-blocking mode

```
/* Socket operation in non-blocking mode */
int32_t  sock1, remain_len, send_len;
int8_t   buffer[1000], *pbuf;
struct sockaddr dest;
int32_t  addr_len;

/*… sock1 was set to non-blocking mode (O_NONBLOCK) */
/*… sock1 was created and TCP sessions established … */
pbuf = &buffer[0];
remain_len = 1000;
/* set the destination addr and len */

/* Call sendto() API */
send_len = sendto(sock1, pbuf, remain_len, 0, &dest, addr_len);
if ((SOCKET_ERROR == send_len) && (EAGAIN == errno))
{
   /* All data in buffer are copied to socket’s transmit internal buffer */
   /* sendto() in non-blocking mode is accepted! */
   remain_len = 0; // Clear remain_len
}
else
{
   /* Handle error process */
}
```

Special Notes

ノンブロッキングモードでは、sendto()は、ソケットの送信バッファに転送したバイト数を戻り値として戻します。この時点では実際にデータは送信されていません。もし送信バッファサイズ(BSD_SND_BUFSZ)より大きなサイズのデータ長を指定した場合、SOCKET_ERROR が戻り、errno = ENOBUFS となります。select()を使用してデータが送信されたことと、新しい送信が可能になったことを確認してください。

3.11 recv()

接続状態のソケットに対してデータを受信します。(TCP)

Format

```
int recv( int sock, const char * buffer, size_t length, int flags )
```

Parameters

sock

ソケット ID

buffer

アプリケーションデータを受信するためのバッファ

length

バッファの長さをバイト長で指定してください。

flags

メッセージフラグです。未実装の引数です。0 を指定してください。

Return Values

SOCKET_ERROR	*処理失敗; エラータイプを示す errno をチェックしてください。*
E_PAR	*パラメータエラー*
Positive Value	*処理成功、受信したデータ長が戻ります。*
0	*処理成功、接続をクローズしました。*

Error Types

EOPNOTSUPP	*socket 引数はフラグでセットされた値の 1 つまたは複数をサポートしないソケットに関連付けられています。*
EPROTONOSUPPORT	*プロトコルはアドレスファミリーもしくは実装でサポートされません。*
ENOTSOCK	*sock 引数はソケットを参照していません。*
ENOBUFS	*システムで利用可能なリソースが不足しています。*
ECONNRESET	*接続先から強制的に接続をクローズされました。*
ENOTCONN	*ソケットは接続されていません。*
EAGAIN	*ソケットファイル記述子に O_NONBLOCK が設定されており、かつ、待ち時間なしに読み出せるデータが受信ウインドウに格納されていません。*
E_QOVR	*2 つ以上の要求が同時に同じソケット記述子で発行されています。*

Properties

Prototyped in r_socket_rx_if.h.

Description

ソケットに受信のあったデータを取り出します。 (TCP)

指定される値は SOCK_STREAM である必要があります。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Example 1: recv() API operation in blocking mode

```
   /* Socket operation in blocking mode */
   int32_t  sock1, remain_len, send_len;
   uint8_t  buffer[1000];
   uint16_t rcvLen;

   /*… sock1 was created and TCP sessions established … */
   /* Call recv() API */
   rcvLen = recv(sock1, buffer, 1000, 0); //API only returns when data is
available on receive window to be read or error occurred.
   if (SOCKET_ERROR == rcvLen)
   {
      /* Handle error or close process */
   }
   else
   {
      /* Data is available to be read */
   }
```

Example 2: 非ブロッキングモードでの **recv () API** 操作

```
   /* Socket operation in non-blocking mode */
   int32_t  sock1, remain_len, send_len;
   uint8_t  buffer[1000];
   uint16_t rcvLen;

   /*… sock1 was set to non-blocking mode (O_NONBLOCK)*/
   /*… sock1 was created and TCP sessions established … */
   /* Call recv() API */
   /* If the socket's receive internal buffer has data,
   this API will copy data to user's buffer and then
   return the actually copied data size.
   Otherwise, it will return SOCKET_ERROR immediately and
   the read request will be accepted to wait for incoming data */
   rcvLen = recv(sock1, buffer, 1000, 0);
   if (rcvLen <= 0)
   {
      if ((SOCKET_ERROR == rcvLen)&&(EAGAIN == errno))
      {
            /* recv() non-blocking is accepted! */
      }
      else
      {
            /* Handle error process */
      }
   }
   else
   {
      /* Data is available in socket's receive internal buffer to be read */
   }
```

Special Notes

受信毎に、実際に受信したデータ長を確認してください。

3.12 recvfrom()

データグラムタイプのソケット(UDP)に対しデータ受信を行います。

Format

```
   int recvfrom( int sock, const char * buffer, size_t length, int flags, struct
sockaddr * from, int * fromlen )
```

Parameters

sock

ソケット ID

buffer

アプリケーションデータを受信するためのバッファ

length

バッファの長さをバイト長で指定してください。

flags

メッセージフラグです。未実装の引数です。0 を指定してください。

from

sockaddr 構造体へのポインタです。構造体には通信相手のアドレス情報(IP アドレスとポート番号)が格納されます。ユーザが値を格納する必要はありません。

fromlen

sockaddr 構造体のサイズが戻ります。

Return Values

SOCKET_ERROR	*処理失敗; エラータイプを示す errno をチェックしてください。*
E_PAR	*パラメータエラー*
Positive Value	*処理成功、受信データ長が戻ります。*

Error Types

EOPNOTSUPP	*socket 引数はフラグでセットされた値の 1 つまたは複数をサポートしないソケットに関連付けられています。*
ENOTSOCK	*sock 引数はソケットを参照していません。*
EADDRNOTAVAIL	*指定されたローカルアドレスは利用可能ではありません。*
ENOBUFS	*システムで利用可能なリソースが不足しています。*
ENOTCONN	*ソケットは接続されていません。*
ECONNRESET	*接続先から強制的に接続をクローズされました。*
EAGAIN	*ソケットファイル記述子に O_NONBLOCK が設定されており、かつ、待ち時間なしに読み出せるデータが受信ウインドウに格納されていません*
EINVAL	*fromlen 引数はアドレスファミリに対して、有効な長さではありません。*

Properties

Prototyped in r_socket_rx_if.h.

Description

データグラムタイプのソケット(UDP)に対しデータ受信を行います。

指定される値は SOCK_DGRAM である必要があります。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Example 1: recvfrom() operation in blocking mode

```
/* Socket operation in blocking mode */
int32_t  sock1, rcvLen;
uint8_t  buffer[1000];
struct sockaddr dest;
int32_t  addr_len;

/*… sock1 was created and TCP sessions established … */
/* Call recvfrom() API */
   rcvLen = recvfrom( sock1, buffer, 1000, 0, &dest, &addr_len);
```

Example 2: recvfrom() operation in non-blocking mode

```
/* Socket operation in non-blocking mode */
int32_t  sock1, rcvLen;
uint8_t  buffer[1000];
struct sockaddr dest;
int32_t  addr_len;

/*… sock1 was set to non-blocking mode (O_NONBLOCK) */
/*… sock1 was created and TCP sessions established … */
/* Call recvfrom() API */
   rcvLen = recvfrom( sock1, buffer, 1000, 0, &dest, &addr_len);
if (rcvLen <= 0)
{
   if ((SOCKET_ERROR == rcvLen)&&(EAGAIN == errno))
   {
         /* recvfrom() non-blocking is accepted! */
   }
   else
   {
         /* Handle error process */
   }
}
else
{
   /* Data is available in socket’s receive internal buffer to be read */
}
```

Special Notes

受信毎に実際の受信データ長を確認してください。

struct sockaddr*from 構造体に格納された送信者 IP アドレスとポート番号に従ったデータ処理を行ってください。また、sockaddr 構造体は送信元 IP アドレスと送信元ポート番号を提供します。

3.13 closesocket()

ソケットをクローズします。

Format

```
int closesocket( int sock )
```

Parameters

sock

Socket ID

Return Values

SOCKET_ERROR	*処理失敗; エラーのタイプを示すために、errno をチェックしてください。*
E_PAR	*パラメータエラー*
E_OK	*処理成功*

Error Types

ENOTCONN	*ソケットは接続されない。*
ENOTSOCK	*ソックス議論によりソケットは参照されます。*

Properties

Prototyped in r_socket_rx_if.h.

Description

ソケットをクローズします。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Special Notes:

この API は T4 のブロッキング機能を使用します。TCP ソケットを閉じる際、T4 の全てのイベントはキャンセルされている必要が有ります。

また、この API は完了するまでに最大 100 ミリ秒の時間を要する場合があります。

この API を呼び出す前にすべてのデータ送信が完了していることを確認してください。

これらの注意事項は TCP ソケットに限られた内容です。UDP ソケットをクローズする場合は特別なハンドシェイクは必要有りません。UDP の接続は通信ごとに毎回クローズされます。

3.14 fcntl()

本関数は、既存ソケットのプロパティを変更します。

Format

```
int fcntl( int sock, int command, int flags )
```

Parameters

sock

ソケット ID

command

F_GETFL: sock 引数で指定されたソケットのタイムアウト値を取得します。

F_SETFL: sock 引数によって指定されたソケットに、タイムアウト値（ブロッキングもしくは非ブロッキング) をセットします。

その他:　無効

flags

タイムアウト値を設定します。 O_NONBLOCK と O_BLOCK のみがサポートされています。

Return Values

SOCKET_ERROR	*処理失敗; エラータイプを示す errno をチェックしてください。*
E_PAR	*パラメータエラー*
E_OK	*設定コマンド操作の成功*

Error Types

ENOTSOCK	*sock 引数はソケットを参照していません。*
EINVAL	*不正な入力パラメータまたはソケットがまだ作成されていません。*

Properties

Prototyped in r_socket_rx_if.h

Description

本関数は、既存ソケットのタイムアウト値を変更します。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Example

```
int32_t  sock1, err;
sock1 = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP);
if( sock1 == SOCKET_ERROR )
{
   /*… check errno and proceed with error handling …*/
}
/* Set socket to non-blocking mode */
err = fcntl(sock1, F_SETFL, O_NONBLOCK);
```

Special Notes:

いずれかのソケットをノンブロッキングモードに設定した場合には、ソケット API を複数タスクから同時に呼び出さないでください。

3.15 select()

本関数は、ソケットのセットに対して、読み込み及び書き込みの準備ができているかどうかをチェックします。 他の場合では、保留中の例外が報告されます。

Format

```
   int select( int nfds, fd_set *p_readfds, fd_set *p_writefds, fd_set
*p_errorfds, struct timeval *timeout )
```

Parameters

nfds

各セットの最初の nfds 個の記述子を調べます。

p_readfds

読み取り準備ができているかどうかをチェックされるべき記述子のセット。 セットしない場合は NULL を設定してください。

p_writefds

書き込み準備ができているかどうかをチェックされるべき記述子のセット。 セットしない場合は NULL を設定してください。

p_errorfds

例外条件がないかどうかチェックされるべき記述子のセット。 セットしない場合は NULL を設定してください。

timeout

タイムアウト値を設定します。NULL を設定した場合、読み取り、書き込みの準備および例外条件が発生するまで本関数を終了しません。

Return Values

SOCKET_ERROR	*処理失敗; エラーのタイプを示すために、errno をチェックして下さい。*
E_PAR	*パラメータエラー*
Positive value	*処理成功。すべての出力セットにおけるソケット記述子の読み込み、書き込み、保留中のエラーの総数。* *p_readfds と p_writefds と p_errorfds が更新されます。*

Error Types

なし

Properties

Prototyped in r_socket_rx_if.h

Description

チェックされるべきソケットのリストを与えます。 各ソケットに対して、読み込み、書き込みの準備ができているか、保留の例外がある場合、同じポインタを通してそれらを返します。

fd_set は 32 ビットの unsigned 型整数値です。

fd_set 型のファイルディスクリプタを操作するためには、FD_SET、FD_CLR、FD_ISSET、FD_ZERO、および FD_ISZERO を使用してください。

FD_SET(fd, fdsetp)は fdsetp によって指定されたセットに、ファイルディスクリプタ、FD を追加します。

FD_CLR(fd、fdsetp)は fdsetp によって指定されたセットから、ファイルディスクリプタ、FD を削除します。FD_ISSET(fd、fdsetp)ファイルディスクリプタが、fd、fdsetp によって指さセットのメンバーである場合、非ゼロ、そうでなければゼロを返します。FD_ZERO（fdsetp）は fdsetp によって示された記述子セットをゼロ初期化します。fd_set は最大で MAX_BSD_SOCKET 個の要素が含まれているものとします。

Reentrant

あり(リアルタイム OS 使用時(SOCKET_IF_USE_SEMP が 1 のとき))

Example

```
int32_t  sock1, child_sock, err;
struct sockaddr_in  serveraddr;
struct sockaddr    clientaddr;
int              clientlen;
fd_set nfds, readfds, writefds, errorfds, rdtestfds, wrtestfds, errtestfds;

/* Create socket */
sock1 = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (SOCKET_ERROR != sock1)
{
   nfds = sock1 + 1;
   FD_SET(sock1, &readfds);
   FD_SET(sock1, &writefds);
   FD_SET(sock1, &errorfds);
}
…
/*...sock1 was set to non-blocking mode */
/*   sock1 was bound, listened */
……
/* Make a connection */
child_sock = accept(sock1, &clientaddr, &clientlen);
if ((SOCKET_ERROR == child_sock) && (EAGAIN == errno)
{
   /* Non-blocking accept() is accepted! */
}
else
{
   closesocket(sock1);
}
...
/* Do something else users want */
...
while(1)
{
```

```
      FD_COPY(&readfds, &rdtestfds);
      FD_COPY(&writefds, &wrtestfds);
      FD_COPY(&errorfds, &errtestfds);
      select(nfds, &rdtestfds, &wrtestfds, &errtestfds, NULL);
      if (FD_ISSET(sock1, &rdtestfds))
      {
            /* The connection has been established */
            /* Be able to start receiving data from client */
            ……
      }
      if (FD_ISSET(sock1, &wrtestfds))
      {
            /* Be able to write data to client */
            ……
      }
      if (FD_ISSET(sock1, &errtestfds))
      {
            /* Either error occurred or sock1 has been closed completely */
            /* Handle the corresponding processes */
            ……
      }
   }
```

4. ユーザインタフェース関数

4.1 概要

表 6 ユーザインタフェース関数一覧

Function	Description
r_socket_task_switch()	API 完了待ち
r_socket_task_switch_select()	select()関数専用処理完了待ち
r_socket_sem_lock()	
r_socket_sem_release()	

4.2 r_socket_task_switch()

ソケット API の処理完了待ち

Format

```
void r_socket_task_switch(int sock)
```

Parameters

sock

ソケット ID

Return Values

None.

Properties

Prototyped in r_socket_rx_if.h.

Description

ソケット API モジュールは、ブロッキングモードで各 API(connect()、accept()、send()、sendto()、recv()、recvfrom())を実行した場合、本関数を繰り返し呼び出します。また、closesocket()を実行した場合は、ノンブロッキングモード、ブロッキングモード問わず、本関数を繰り返し呼び出します。

ユーザはリアルタイム OS を使用する場合、タスクスイッチができるシステムコール(ITRON の場合、dly_tsk())を呼出してください。リアルタイム OS を使用しない場合は、何も呼び出さないでください。

Example

```
void r_socket_task_switch(int sock)
{
   /* If user uses "Real time OS", user may define "sleep task" here. */
#if BSP_CFG_RTOS_USED == 0    // Non-OS
#elif BSP_CFG_RTOS_USED == 1    // FreeRTOS
   vTaskDelay(2 / portTICK_RATE_MS);
#elif BSP_CFG_RTOS_USED == 2    // SEGGER embOS
#elif BSP_CFG_RTOS_USED == 3    // Micrium MicroC/OS
#elif BSP_CFG_RTOS_USED == 4    // Renesas RI600V4 & RI600PX
   dly_tsk(2);
#endif
}
```

4.3 r_socket_task_switch_select()

select()関数の処理完了待ち

Format

```
void r_socket_task_switch_select(void)
```

Parameters

None.

Return Values

None.

Properties

Prototyped in r_socket_rx_if.h.

Description

ソケット API モジュールは、ユーザが select()を実行したとき、本関数を繰り返し呼び出します。ユーザはリアルタイム OS を使用する場合、タスクスイッチができるシステムコール(ITRON の場合、dly_tsk())を呼出してください。リアルタイム OS を使用しない場合は、何も呼び出さないでください。

Example

```
void r_socket_task_switch_select(void)
{
#if BSP_CFG_RTOS_USED == 0   // Non-OS
#elif BSP_CFG_RTOS_USED == 1   // FreeRTOS
   vTaskDelay(2 / portTICK_RATE_MS);

#elif BSP_CFG_RTOS_USED == 2   // SEGGER embOS
#elif BSP_CFG_RTOS_USED == 3   // Micrium MicroC/OS
#elif BSP_CFG_RTOS_USED == 4   // Renesas RI600V4 & RI600PX
   dly_tsk(2);
#endif
}
```

4.4 r_socket_sem_lock()

セマフォのロック

Format

```
int r_socket_sem_lock(void)
```

Parameters

None.

Return Values

SOCKET_ERROR	*処理失敗*
E_OK	*処理成功*

Properties

Prototyped in r_socket_rx_if.h.

Description

本関数は SOCKET_IF_USE_SEMP=1 の場合に呼び出されます。

リアルタイム OS 使用時はセマフォを獲得する関数を呼びだしてください。

Example

```
#if BSP_CFG_RTOS_USED == 1    // FreeRTOS
extern xSemaphoreHandle r_socket_semaphore;
#elif BSP_CFG_RTOS_USED == 4    // Renesas RI600V4 & RI600PX
extern ID r_socket_semaphore;
#endif

int r_socket_sem_lock(void)
{
    int retcode;
    retcode = E_OK;
#if BSP_CFG_RTOS_USED == 0    // Non-OS
#elif BSP_CFG_RTOS_USED == 1    // FreeRTOS
    if (pdTRUE != xSemaphoreTake(r_socket_semaphore, portMAX_DELAY))
    {
        retcode = SOXKER_ERROR;
    }
#elif BSP_CFG_RTOS_USED == 2    // SEGGER embOS
#elif BSP_CFG_RTOS_USED == 3    // Micrium MicroC/OS
#elif BSP_CFG_RTOS_USED == 4    // Renesas RI600V4 & RI600PX
    if (E_OK != pol_sem ( r_socket_semaphore ))
    {
        retcode = SOXKER_ERROR;
    }
#endif
    return retcode;
}
```

4.5 r_socket_sem_release()

セマフォの解放

Format

```
int r_socket_sem_release(void)
```

Parameters

None.

Return Values

SOCKET_ERROR	*処理失敗*
E_OK	*処理成功*

Properties

Prototyped in r_socket_rx_if.h.

Description

本関数は SOCKET_IF_USE_SEMP=1 の場合に呼び出されます。

リアルタイム OS 使用時はセマフォを解放する関数を呼びだしてください。

Example

```
#if BSP_CFG_RTOS_USED == 1    // FreeRTOS
extern xSemaphoreHandle r_socket_semaphore;
#elif BSP_CFG_RTOS_USED == 4    // Renesas RI600V4 & RI600PX
extern ID r_socket_semaphore;
#endif

int r_socket_sem_release(void)
{
    int retcode;
    retcode = E_OK;
#if BSP_CFG_RTOS_USED == 0    // Non-OS
#elif BSP_CFG_RTOS_USED == 1    // FreeRTOS
    if (pdTRUE != xSemaphoreGive(r_socket_semaphore))
    {
        retcode = SOXKER_ERROR;
    }
#elif BSP_CFG_RTOS_USED == 2    // SEGGER embOS
#elif BSP_CFG_RTOS_USED == 3    // Micrium MicroC/OS
#elif BSP_CFG_RTOS_USED == 4    // Renesas RI600V4 & RI600PX
    if (E_OK != sig_sem ( r_socket_semaphore ))
    {
        retcode = SOXKER_ERROR;
    }
#endif
    return retcode;
}
```

5. 注意事項

5.1 複数 Ethernet チャネル対応について

本モジュールでは 1 チャネルのみ対応しています。

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.31	2016.10.01	--	FIT 用 xml ファイルを更新しました。 ユーザインタフェース関数を追加しました。 USE_BSD_NON_BLOCKING マクロを削除しました。 FD_SETSIZE マクロを削除しました。 SOCKET_TCP_WINSIZE マクロを追加しました R_SOCKET_Init()の API 名を R_SOCKET_Open()に変更しました R_SOCKET_Close()を追加しました。 Ether-2 チャネルのサポートを非対応にしました。 本資料に 4 章と 5 章を追加しました。
1.30	2015.09.15	--	fcntl(), select()を追加しました。 errno を各 API に追加しました。 send/sendto/accept の説明文を更新しました。
1.22	2015.02.12	--	ソースコードを修正しました。
1.21	2015.01.31	--	FIT モジュール名を変更しました。 RX71M に対応しました。
1.20	2014.07.01	--	Ether-2 チャネルをサポートしました。
1.10	2014.04.01	--	リビジョンをソフトウェアバージョンに合わせて変更。
1.00	--	--	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子
   （または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。
注2. 本資料において使用されている「当社製品」とは、注１において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/