RX Family

Embedded TCP/IP M3S-T4-Tiny Socket API Module

Firmware Integration Technology

R20AN0296EJ0132
Rev.1.32
Feb 01, 2019

Introduction

This is the Socket API FIT Module for Embedded TCP/IP M3S-T4-Tiny (Hereafter T4).

T4 has APIs corresponds ITRON TCP/IP. Many regions, and many people like a network APIs are “Socket APIs”. So, many people will be able to develop T4 application, we prepared socket APIs for T4. User can use socket APIs adding this module to T4 system.

For about T4, please refer to the following URL.

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

Socket APIs and T4 are provided as FIT Module. Please refer to the URL to understand FIT outline.

FIT: Firmware Integration Technology.

https://www.renesas.com/en-us/solutions/rx-applications/fit.html

This figure shows 2 cases of T4 software stack.

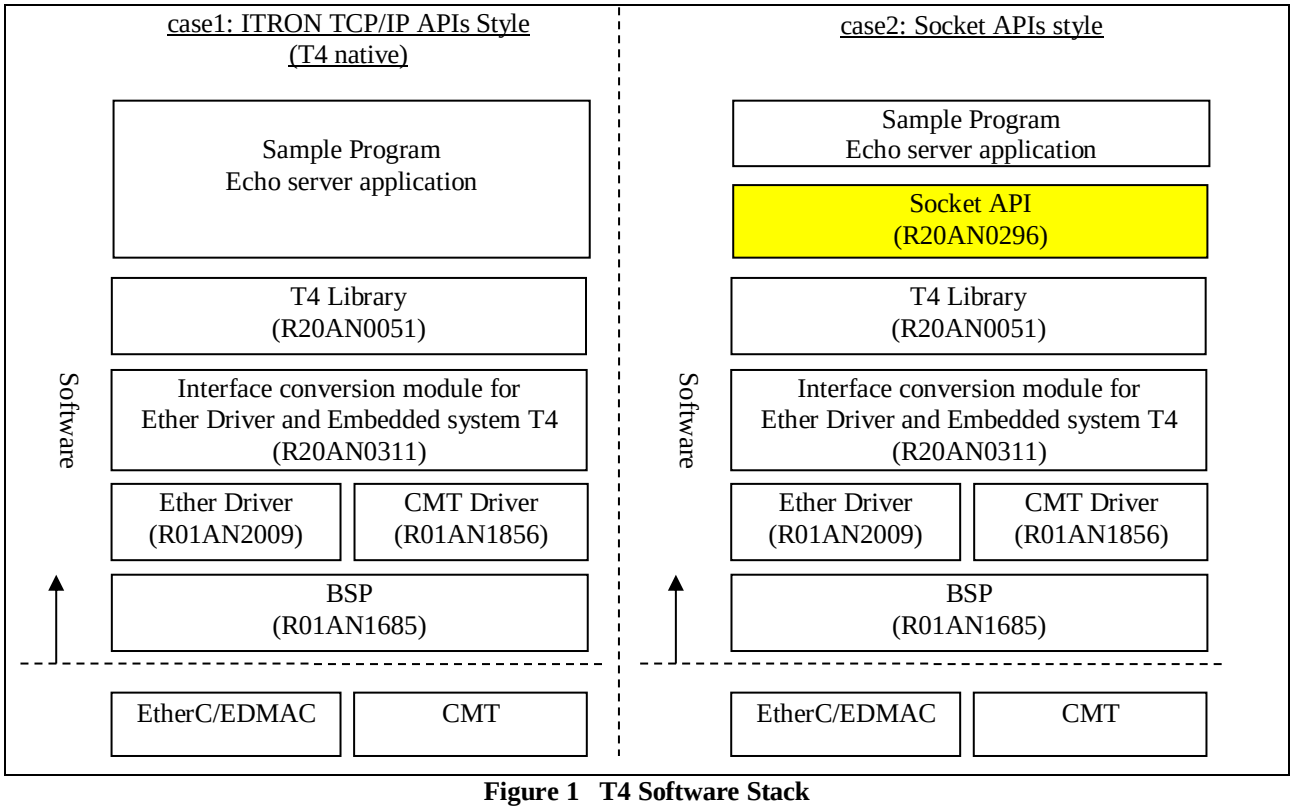

Figure 1 T4 Software Stack

Notice:

This socket API is easily implementation and this socket API provides basic functions only. It is impossible to port the apache etc using generic socket API applications to this module with T4.

Target Device

RX Family

Contents

1. Overview

1.1 Mapping of Socket APIs to T4 APIs

Table 1 below provides the mapping list of socket APIs to T4 APIs.

Table 1 Mapping list of socket APIs to T4 APIs

No	Function Descriptions	Socket APIs	T4 API Mapped To
1	Open the socket APIs.	R_SOCKET_Open()	tcpudp_get_ramsize() tcpudp_open()
2	Close the socket APIs	R_SOCKET_Close()	tcpudp_close()
3	Creates a new socket of a certain socket type, identified by an integer number, and allocates system resources to it.	socket()	get_random_number()
4	bind() can set the local port number to use in accept()	bind()	-
5	Used on the client side, and assigns a free local port number to a socket. In case of a TCP socket, it causes an attempt to establish a new TCP connection. Use the fixed IP address and port number when UDP is selected.	connect()	tcp_con_cep() get_random_number()
6	Used on the server side, and causes a bound TCP socket to enter listening state.	listen()	tcp_acp_cep()
7	Used on the server side. It accepts a responds to incoming attempt to create a new TCP connection from a remote client.	accept()	tcp_acp_cep() tcp_rcv_dat()
8	Writes data to the socket from buffer.	send()	tcp_can_cep() tcp_snd_dat()
9	Writes data the remote host specified into buffer. The socket must be a SOCK_DGRAM (UDP) socket	sendto()	udp_snd_dat()
10	Reads data from the socket into buffer.	recv()	tcp_rcv_dat()
11	Reads data from the remote host specified by fromAddr into buffer. The socket must be a SOCK_DGRAM (UDP) socket.	recvfrom()	
12	Finish sending.	-	tcp_sht_cep()
13	Closes an existing socket	closesocket()	tcp_can_cep() tcp_cls_cep() udp_can_cep()
14	Modify a socket	fcntl()	-
15	Synchronous I/O multiplexing for a socket	select()	tcpudp_get_time()

2. API Information

This API adheres to the Renesas API naming standards.

2.1 Hardware Requirements

None

2.2 Software Requirements

This FIT Module is dependent upon the following packages:

- r_t4_rx

- r_t4_driver_rx

2.3 Supported Toolchains

This driver is tested and works with the following toolchain:

- Renesas RX Toolchain v.2.05.00

2.4 Header Files

All API calls and their supporting interface definitions are located in r_socket_rx_if.h.

2.5 Integer Types

This project uses ANSI C99 "Exact width integer types" in order to make the code clearer and more portable. These types are defined in *stdint.h.*

2.6 Configuration Overview

The configuration options in this module are specified in r_socket_rx_config.h. The option names and setting values are listed in the table below.

Table 2 configuration options

Configuration options in *r_socket_rx_config.h*	
#define MAX_UDP_CCEP **- Default value = 4**	The number of UDP communication end-point allocated for T4. Please select a suitable number for your system.
#define MAX_TCP_CCEP **- Default value = 4**	The number of TCP communication end-point allocated for T4. Please select a suitable number for your system. Please set the 2 or more to the MAX_TCP_CCEP.
#define MAX_TCP_CREP **- Default value = MAX_TCP_CCEP**	The number of TCP reception end-point allocated for T4. Typically we allocate a number equal to the number of TCP communication end-point.
#define SOCKET_TCP_WINSIZE **- Default value = 1460**	The window size for T4.
#define TCPUDP_WORK **- Default value = 7200**	Size of the work area used by T4. Work area size is dependent on the number and type of sockets allocated. The size of the work area can be determined by this T4 API "*tcpudp_get_ramsize()*". Default value is 7200 bytes when MAX_TCP_CCEP = 4 and MAX_UDP_CCEP = 4.
#define TOTAL_BSD_SOCKET **- Default value = (MAX_UDP_CCEP+ MAX_TCP_CCEP)**	The total number of sockets that can be used. This parameter corresponds to the total number of T4 communications endpoints defined in structure "*tcp_ccep[]*" and "*udp_ccep[]*".
#define SOCKET_IF_USE_SEMP **- Default value = 0**	If a suitable locking mechanism or semaphore is available, please set to 1. This will protect critical sections in the *socket()* API from concurrent function call.
#define R_SOCKET_PAR_CHECK **- Default value = 1**	#undef this if you want to skip parameter checking in all of the socket APIs.
#define BSD_RCV_BUFSZ **- Default value = 1460**	Size of the receive buffer used to store data received by socket.
#define BSD_SND_BUFSZ **- Default value = 1460**	Size of the transmit buffer used to store data transmitted by socket.

2.7 API Data Structure

This section details the data structures that are used with the wrapper's API functions.

```
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;
```

2.8 Return Values

This shows the different values API functions can return. These definitions are all found in r_socket_rx_if.h.

```
/**** 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 Error Codes

This shows all error codes used in socket APIs.

Table 3 Error Codes

Error Code	Value	Significance
ENFILE	23	No more file descriptors are available
EAGAIN	11	The non-blocking request has been accepted
EINPROGRESS	150	The connection cannot be connected immediately
EALREADY	37	The requested socket is in use
ENOTSOCK	38	No valid socket to refer
EDESTADDRREQ	39	Socket is not bound to local address
EPROTOTYPE	41	Socket type is not supported
EPROTONOSUPPORT	43	Protocol is not supported
EOPNOTSUPP	45	The socket is in listening mode and cannot be connected
EAFNOSUPPORT	47	Address family is not supported
ECONNRESET	54	The connection was forcibly closed by a peer
EISCONN	56	The specified socket is already connected
ENOTCONN	57	The specified socket is not connected

2.10 Adding Driver to Your Project

The driver must be added to an existing e²studio project. It is the best to use the e²studio FIT plugin to add the driver to your project as that will automatically update the include file paths for you. Alternatively, the driver can be imported from the archive that accompanies this application note and manually added by following these steps:

1. This application note is distributed with a zip file package that includes the Embedded TCP/IP M3S-T4-Tiny Socket API module in its own folder *r_socket.*
2. Unzip the package into the location of your choice.
3. In a file browser window, browse to the directory where you unzipped the distribution package and locate the *r_socket* folder.
4. Open your e²studio workspace.
5. In the e²studio project explorer window, select the project that you want to add the socket module to.
6. Drag and drop the *r_socket* folder from the browser window (or copy/paste) into your e²studio project at the top level of the project
7. Update the source search/include paths for your project by adding the paths to the module files:
    a. Navigate to the "*Add directory path*" control:
        i. *'project name'->properties->C/C++ Build->Settings->Compiler->Source -Add* (*green +icon*)
    b. Add the following paths:
        i. *"${workspace_loc:/${ProjName}/r_socket}"*
        ii. *"${workspace_loc:/${ProjName}/r_socket/src}*"

Whether you used the plug-in or manually added the package to your project, it is necessary to configure the driver for your application.

8. Locate the *r_socket_config_reference.h* file in the *r_socket/ref/* source folder in your project and copy it to your project's *r_config* folder.
9. Change the name of the copy in the *r_config* folder to *r_socket_config.h.*
10. Make the required configuration settings by editing the copied *r_socket_config.h* file. Please refer to **Chapter 2.6 Configuration Overview**.

3. API Functions

3.1 Summary

Table 4 List of API functions supported by Socket Module

Function	Description
R_SOCKET_Open()	Initialize all socket structures.
R_SOCKET_Close()	Close socket module.
socket()	Create a new socket
bind()	Bind socket to local address
connect()	Request a connection to server side
listen()	Place socket in listening state
accept()	Accept a connection from client side
send()	Send data to stream socket
sendto()	Send data to datagram socket
recv()	Receive data from stream socket
recvfrom()	Receive data from datagram socket
closesocket()	Close an existing socket
fcntl()	Modify time-out value of socket
select()	Synchronize I/O multiplexing

3.2 R_SOCKET_Open()

Initialize the socket structure to a known initial value.

Format

```
void R_SOCKET_Open( void )
```

Parameters

None

Return Values

None

Error Types

None

Properties

Prototyped in r_socket_rx_if.h

Description

Initialize the socket structure to a known initial value. It also sets the *rbufsz* of the underlying CCEP, the T4 communication end-point data structure.And this API is calling *tcpudp_open*(). *tcpudp_open*() function uses the parameters in CCEP end point (such as *rbufsz*) to allocate reception buffers within the *tcpudp_work*[] work RAM.

Reentrant

No

Examples

```
R_SOCKET_Open();
```

Special Notes

This API initializes the tcp_ccep[], the data structure for each TCP communication end-point used in T4. In particular, the size of the receive buffer, rbufsz, corresponding to each TCP end point. T4's tcpudp_open() API uses this parameter to assign buffer addresses from T4's working memory, tcpudp_work[]. And this API calling T4's tcpdudp_open(). And initialize for network layer API, lan_open() should be called with this API. Please call lan_open(), R_SOCKET_Open() in this sequence.

3.3 R_SOCKET_Close()

Close the socket APIs.

Format

```
void R_SOCKET_Close( void )
```

Parameters

None

Return Values

None

Error Types

None

Properties

Prototyped in r_socket_rx_if.h

Description

Reentrant

No

Examples

```
R_SOCKET_Close();
```

Special Notes

3.4 socket()

This function creates a new socket.

Format

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

Parameters

domain

AF_INET are acceptable. SOCKET_ERROR is returned when other values are specified.

type

SOCK_STREAM: a TCP Socket is created.
SOCK_DGRAM: a UDP Socket is created.

protocol

Set IP protocol to “IPPROTO_UDP” when type is SOCK_DGRAM
or set to “IPPROTO_TCP” when type is SOCK_STREAM.

Return Values

SOCKET_ERROR	*Operation failed; check errno to indicate the type of error*
E_PAR	*Parameter Error*
0 or Positive value	*Operation successfully, and return socket ID.*

Error Types

EAFNOSUPPORT	*The specified address family is not supported.*
EPROTOTYPE	*The socket type is not supported by the protocol.*
ENFILE	*No more sockets are available.*
EPROTONOSUPPORT	*The protocol is not supported by either the address family or the implementation.*

Properties

Prototyped in *r_socket_rx_if.h*

Description

This function creates a new socket.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

Example

```
int32_t  sock1;

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

Special Notes

Socket number {0 … *MAX_TCP_CCEP*-1} are reserved for TCP type while those in the range {*MAX_TCP_CCEP*…(*MAX_TCP_CCEP*+*MAX_UDP_CCEP*-1)} are for UDP type.

A locking mechanism must be provided to protect critical sections in systems where concurrent *socket*() call cannot be avoided.

3.5 bind()

The bind function assigns a name to an unnamed socket.

Format

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

Parameters

sock

Socket identifier

name

Pointer to the sockaddr structure containing the local address of the socket

namelen

Length of the sockaddr structure

Return Values

SOCKET_ERROR	*Operation failed; check errno to indicate the type of error*
E_PAR	*Parameter Error*
E_OK	*Operation successful*

Error Types

ENOTSOCK	*The sock argument does not refer to a socket.*
EADDRNOTAVAIL	*The specified local address is not available.*
EINVAL	*Socket is already bound or the protocol doesn't require binding or the socket has been shut down.*
EPROTONOSUPPORT	*The protocol is not supported by either the address family or the implementation.*

Properties

Prototyped in *r_socket_rx_if.h*

Description

The bind function assigns a name to an unnamed socket. The name refers to an IP address and port number.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

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)(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()

This function connects to a host.

Format

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

Parameters

sock

Socket identifier

name

Pointer to the sockaddr structure containing the remote host's IP address and port number

namelen

Length of the sockaddr structure

Return Values

SOCKET_ERROR	*Operation failed, check errno to indicate the type of error*
E_PAR	*Parameter Error*
E_OK	*Operation successful*

Error Types

ENOTSOCK	*The sock argument does not refer to a socket.*
EADDRNOTAVAIL	*The specified local address is not available.*
EALREADY	*A connection request is already in progress for the specified socket.*
EISCONN	*The specified socket is connection-mode and is already connected.*
EOPNOTSUPP	*The socket is not in the right state (listening etc.,) and cannot be connected.*
EINVAL	*The address length is not a valid length for the address family or invalid address family in the sockaddr structure.*
EINPROGRESS	*O_NONBLOCK is set for timeout. The request is being performed asynchronously.*
ETIMEDOUT	*The attempt to connect timed out before a connection was made.*
EPROTONOSUPPORT	*The protocol is not supported by either the address family or the implementation.*

Properties

Prototyped in *r_socket_rx_if.h*

Description

This function initiates a connection request to a host by sending it a TCP SYN signal.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

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 = 0x0800A8C0; // 192.168.0.8
serveraddr.sin_port = (unsigned short)1024;

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

Special Notes

In socket non-blocking mode, TMO_NBLK is set for tmout argument of the BSD socket's structure: When connect() API is called, if the connection cannot be established immediately, the connect() API will return SOCKET_ERROR and set errno to EINPROGRESS. However the connection request will not be aborted, the connection will be established asynchronously. Before the connection is established, a subsequent calls to connect() for the same socket will be failed and set errno to EALREADY.

3.7 listen()

The listen function sets the specified socket to a listening mode. After transiting to listening mode, this specified socket will wait for an incoming client in case of non-blocking mode.

Format

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

Parameters

sock

Socket identifier

backlog

Maximum number of connection requests that can be queued (not in use). Please set to 1.

Return Values

SOCKET_ERROR	*Operation failed; check errno to indicate the type of error*
E_PAR	*Parameter Error*
E_OK	*Operation successful*

Error Types

ENOTSOCK	*The sock argument does not refer to a socket.*
ENOBUFS	*Insufficient resources are available in the system.*
EINVAL	*The socket has been shut down.*
EDESTADDRREQ	*The socket is not bound to a local address and the protocol does not support listening on an unbound socket.*
EOPNOTSUPP	*The socket is not in the right state (listening etc.,) and cannot be connected.*
ENFILE	*No more sockets are available.*

Properties

Prototyped in r_socket_rx_if.h

Description

The listen function sets the specified socket to enter listening mode.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

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

In non-blocking mode, another socket is acquired internally and is switched to be a BSD_CONNECTING socket. This socket is placed on standby for connecting. If there are no spare socket, SOCKET_ERROR is returned with errno = ENFILE.

3.8 accept()

The accept function is used to accept a connection request queued for a listening socket.

Format

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

Parameters

sock

Socket identifier

address

Pointer to the sockaddr structure that will receive the connecting node IP address and port number, user does not need to store the value.

address_len

A value-result parameter and should initially contain the amount of space pointed to by address_len. On return it contains the actual length in bytes of the address_len returned.

Return Values

SOCKET_ERROR	*Operation failed; check errno to indicate the type of error*
E_PAR	*Parameter Error*
Positive Value	*Operation successful, returned Socket identifier*

Error Types

ECONNABORTED	*A connection has been aborted.*
ENOTSOCK	*The sock argument does not refer to a socket.*
EADDRNOTAVAIL	*The specified local address is not available.*
EAGAIN	*O_NONBLOCK is set for the socket file descriptor and no connections are present to be accepted.*
EINVAL	*The socket is not accepting connections.*
EOPNOTSUPP	*The socket type of the specified socket does not support accepting connections.*

Properties

Prototyped in *r_socket_rx_if.h*

Description

The accept function is used to accept a connection request queued for a listening socket.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

Example

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

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

Special Notes

In socket non-blocking mode, when *accept*() API is called, if there are no connection to be accepted, the *accept*() API will return *SOCKET_ERROR* immediately with *errno* set to *EAGAIN*. Later use the, *select*() API to verify whether the connection has been established.

If the accepted socket is the same as the original socket, the original socket cannot accept any more connections. To avoid this situation, please prepare in advance the number of available socket is 2 more than the desired number of connections. e..g if 4 connection is desired, prepare 6 sockets, 1 for listening, 1 as standby for connection and the other 4 to accept the connection.

Table 5 : Accept with 4 available sockets (non-blocking mode)

Socket's role	Listener	Standby	Child socket	Remarks
Socket status	BSD_LISTENING	BSD_CONNECTING	BSD_CONNECTED	
Socket() bind(),	0	---	---	Socket #0 is created and bind to a local address and port
listen()	0	1[1]	---	Socket #0 is placed in listening mode and socket #1 is on standby for connection
After 1st accept	0	2[1]	1	Socket #1 is returned as child socket, socket#2 becomes standby for connection
After 2nd accept	0	3[1]	2	Socket #2 is returned as child socket, socket#3 becomes standby for connection
After 3rd accept	0	-1[2]	3	Socket #3 is returned as child socket. As there are no more socket, the standby socket is -1.
When 4th accept	0	-1[2]	SOCKET_ERROR errno = ENFILE	At 4th accept, SOCKET_ERROR is returned with *errno* = ENFILE;
Example of operations:- after some time, socket #2 is closed. When called, the select() will detect an unused socket and raise a "can_read" flag for the listening socket, signifying the user application to issue an accept().				
After 5th accept	0	2[3]	SOCKET_ERROR errno = ENFILE	Socket #2 now becomes the standby socket. The accept API still returns a SOCKET_ERROR. Subsequent accept will return the standby socket (i.e. socket #2).
Close the listener socket #0	---	---	---	When socket #0 is closed, the standby socket (currently #2) will also be closed.

[1] Next available socket, assuming sequence is 1,2,3

[2] All sockets (0,1,2,3) are used. A -1 signifies invalid socket number.

[3] Socket #2 has been closed. It is now available for use as standby socket when user issue the accept() API.

3.9 send()

The function is used to send outgoing data to a connected socket. (TCP)

Format

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

Parameters

sock

Socket identifier

buffer

Application data buffer containing data to transmit

length

Length of data in bytes. Maximum length of data is 0x7FFFH for blocking mode and BSD_SND_BUFSZ for non-blocking mode.

flags

Message flags. Currently this field is not supported and must be 0.

Return Values

SOCKET_ERROR	*Operation failed; check errno to indicate the type of error*
E_PAR	*Parameter Error*
Positive Value	*Operation successful, transmitted data size will be returned.*

Error Types

ENOTCONN	*The socket is not connected.*
ENOTSOCK	*The sock argument does not refer to a socket.*
EADDRNOTAVAIL	*The specified local address is not available.*
ECONNRESET	*A connection was forcibly closed by a peer.*
EOPNOTSUPP	*The socket argument is associated with a socket that does not support one or more of the values set in flags.*
ENOBUFS	*Insufficient resources are available in the system.*
EAGAIN	*O_NONBLOCK is set for the socket file descriptor and no wait for the data is transmitted completely.*
E_QOVR	*Two or more requests are issued on same socket descriptor concurrently.*

Properties

Prototyped in *r_socket_rx_if.h*

Description

The function is used to send outgoing data on a socket of type stream.
The socket type must be SOCK_STREAM.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

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

In socket non-blocking mode, *send*() API will return the number of bytes that is transferred to socket's sending buffer. The actual data may not been transferred. If the length to be sent is greater than the size of send buffer(BSD_SND_BUFSZ), a SOCKET_ERROR is returned with *errno* = ENOBUFS. Please confirm using the select() API that all data has been transfer and that a new transfer can be initiated.

3.10 sendto()

The function is used to send outgoing data on a socket of type datagram only. (UDP)

Format

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

Parameters

sock
Socket identifier
buffer
Application data buffer containing data to transmit
length
Length of data in bytes. Maximum length of data is 0x7FFFH for blocking mode and BSD_SND_BUFSZ for non-blocking mode.
flags
Message flags. Currently this field is not supported and must be 0.
to
Pointer to the sockaddr structure containing the destination address
tolen
Length of the sockaddr structure

Return Values

SOCKET_ERROR	*Operation failed; check errno to indicate the type of error*
E_PAR	*Parameter Error*
Positive Value	*Operation successful, transmitted data size will be returned.*

Error Types

EOPNOTSUPP	*The socket argument is associated with a socket that does not support one or more of the values set in flags.*
ENOTCONN	*The socket is not connected.*
ENOTSOCK	*The sock argument does not refer to a socket.*
EADDRNOTAVAIL	*The specified local address is not available.*
ENOBUFS	*Insufficient resources are available in the system.*
ECONNRESET	*A connection was forcibly closed by a peer.*
EINVAL	*The tolen argument is not a valid length for the address family.*
EAGAIN	*O_NONBLOCK is set for the socket file descriptor and no wait for the data is transmitted completely.*

Properties

Prototyped in *r_socket_rx_if.h*

Description

The function is used to send outgoing data on a socket of type datagram.
The socket type must be SOCK_DGRAM.
The recipient's address and port number must always be supplied.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

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 */
send_len = sendto( sock1, pbuf, remain_len, 0, &dest, addr_len );
```

Example 2: 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 (remain_len == send_len)
{
   /* 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

In socket non-blocking mode, *sendto*() API will return the number of bytes that is transferred to socket's sending buffer. The actual data may not been transferred. If the length to be sent is greater than the size of sending buffer(BSD_SND_BUFSZ), a SOCKET_ERROR is returned with *errno* = ENOBUFS. Please confirm using the select() API that all data has been transfer and that a new transfer can be initiated.

3.11 recv()

The function is used to receive incoming data that has been queued for a socket. (TCP)

Format

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

Parameters

sock
Socket identifier
buffer
Application data receive buffer
length
Buffer length in bytes
flags
Message flags. Currently this field is not supported and must be set to 0.

Return Values

SOCKET_ERROR	*Operation failed; check errno to indicate the type of error*
E_PAR	*Parameter Error*
Positive Value	*Operation successful, received data size will be returned.*
0	*Operation successful, a connection was closed by a peer.*

Error Types

EOPNOTSUPP	*The socket argument is associated with a socket that does not support one or more of the values set in flags.*
EPROTONOSUPPORT	*The protocol is not supported by either the address family or the implementation.*
ENOTSOCK	*The sock argument does not refer to a socket.*
ENOBUFS	*Insufficient resources are available in the system.*
ECONNRESET	*A connection was forcibly closed by a peer.*
ENOTCONN	*The socket is not connected.*
EAGAIN	*O_NONBLOCK is set for the socket file descriptor and no wait for the data is available on receive window to be read.*
E_QOVR	*Two or more requests are issued on same socket descriptor concurrently.*

Properties

Prototyped in *r_socket_rx_if.h*

Description

The function is used to receive incoming data that has been queued for a socket.
The socket type must be SOCK_STREAM.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

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 an error has occurred.
   if (SOCKET_ERROR == rcvLen)
   {
      /* Handle error or close process */
   }
   else
   {
      /* Data is available to be read */
   }
```

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

```
   /* 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 size of copied data.
   Otherwise, it will return SOCKET_ERROR immediately */
   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

Please check the actual number of bytes received.

3.12 recvfrom()

The function is used to receive incoming data that has been queued for a socket of type datagram. (UDP)

Format

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

Parameters

sock
Socket identifier
buffer
Application data receive buffer
length
Buffer length in bytes
flags
Message flags. Currently this field is not supported and must be 0
from
Pointer to the sockaddr structure that will be filled in with the destination address
fromlen
Size of sockaddr structure

Return Values

SOCKET_ERROR	*Operation failed; check errno to indicate the type of error*
E_PAR	*Parameter Error*
Positive Value	*Operation successful, received data size will be returned.*

Error Types

EOPNOTSUPP	*The socket argument is associated with a socket that does not support one or more of the values set in flags.*
ENOTSOCK	*The sock argument does not refer to a socket.*
EADDRNOTAVAIL	*The specified local address is not available.*
ENOBUFS	*Insufficient resources are available in the system.*
ENOTCONN	*The socket is not connected.*
ECONNRESET	*A connection was forcibly closed by a peer.*
EAGAIN	*O_NONBLOCK is set for the socket file descriptor and no wait for the data is available on receive window to be read.*
EINVAL	*The fromlen argument is not a valid length for the address family.*

Properties

Prototyped in r_socket_rx_if.h

Description

The function is used to receive incoming data that has been queued for a socket.
The socket type must be SOCK_DGRAM.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

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

Please check the actual number of bytes received and process the data according to the sender IP address and port number as given in the *struct sockaddr *from* structure.

3.13 closesocket()

The function closes an existing socket.

Format

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

Parameters

sock

Socket identifier

Return Values

SOCKET_ERROR	*Operation failed; check errno to indicate the type of error*
E_PAR	*Parameter Error*
E_OK	*Operation successful*

Error Types

ENOTCONN	*The socket is not connected.*
ENOTSOCK	*The sock argument does not refer to a socket.*
EAGAIN	*O_NONBLOCK is set for the socket file descriptor and no wait for closing socket is processed completely.*

Properties

Prototyped in *r_socket_rx_if.h*

Description

The function closes an existing socket.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

Special Notes

In case this socket API uses the blocking method of T4. When a TCP socket is closed, all outstanding T4 events must be cancelled. And this API might take 100 milliseconds to complete.

Please ensure all data transfer is complete before issuing this *closesocket*() API.
The above remark applies to TCP socket. When closing a UDP socket, no handshake is needed between its partners. A UDP connection can be closed by either side at any time.

3.14 fcntl()

The function modifies the properties of an existing socket.

Format

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

Parameters

sock

Socket identifier

command

F_GETFL: Get the timeout value of the socket specified by sock argument.
F_SETFL: Set the timeout value to blocking or non-blocking for the socket specified by sock argument.
Others: Invalid.

flags

timeout value to be set. O_NONBLOCK and O_BLOCK are supported only.

Return Values

SOCKET_ERROR	*Operation failed; check errno to indicate the type of error*
E_PAR	*Parameter Error*
E_OK	*Set command operation successful*

Error Types

ENOTSOCK	*The sock argument does not refer to a socket.*
EINVAL	*Bad input parameters or the socket has not created yet.*

Properties

Prototyped in *r_socket_rx_if.h*

Description

The function modifies the timeout value of an existing socket.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

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

When a non-blocking mode for some sockets is selected, please ensure that socket APIs are not issued from multiple task concurrently.

3.15 select()

This function checks a set of sockets for their readiness to perform read or write operation. In other cases, exception conditions that are pending will be reported.

Format

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

Parameters

nfds

Examine the first nfds descriptor of each set.

p_readfds

A set of descriptors to be checked for read readiness. Set NULL if no check.

p_writefds

A set of descriptors to be checked for write readiness. Set NULL if no check.

p_errorfds

A set of descriptors to be checked for exception conditions. Set NULL if no check.

timeout

A timeout value to be set. This function will be not finished until occurring event (readable/writeable/ exception) when set the NULL to timeout.

Return Values

SOCKET_ERROR	*Operation failed; check errno to indicate the type of error*
E_PAR	*Parameter Error*
Positive value	*Operation successful. The total number of socket's descriptor ready for writing, reading or error pending in all output sets. p_readfds, p_writefds, p_errorfds are updated.*

Error Types

None

Properties

Prototyped in *r_socket_rx_if.h*

Description

A list(s) of sockets is presented for checking. If any sockets are ready for reading, writing or an exceptions condition is pending, they are returned via the same pointer.

The *fd_set* is a 32bit fixed unsigned variable.

Please use functions FD_SET, FD_CLR, FD_ISSET, FD_ZERO and FD_ISZERO to manipulate file descriptors of type *fd_set*.
FD_SET(*fd*, *fdsetp*) adds the file descriptor, *fd*, to the set pointed to by *fdsetp*.
FD_CLR(*fd*, *fdsetp*) removes the file descriptor, *fd*, from the set pointed to by *fdsetp*.
FD_ISSET(*fd, fdsetp*) shall evaluate to non-zero if the file descriptor, *fd*, is a member of the set pointed to by *fdsetp*, and shall evaluate to zero otherwise.
FD_ZERO(*fdsetp*) shall initialize the descriptor set pointed to by *fdsetp* to the null set. It is assumed the *fd_set* contains MAX_BSD_SOCKET elements.
FD_ISZERO(*fdsetp*) shall verify whether or not all file descriptors in the set are equal to 0.

Reentrant

Yes (When using Realtime OS(When SOCKET_IF_USE_SEMP is defined to 1)

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. User Interface function

Please implement following user interface functions by user.

In case, use with realtime os : Implement referring each Example

In case, use without realtime os : Implement as empty function

4.1 Summary

Table 6 List of User Interface functions supported by Socket Module

Function	Description
r_socket_task_switch()	Initialize all socket structures.
r_socket_task_switch_select()	Create a new socket
r_socket_sem_lock()	
r_socket_sem_release()	

4.2 r_socket_task_switch()

Waiting for socket API process.

Format

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

Parameters

sock

socket ID

Return Values

None.

Properties

Prototyped in r_socket_rx_if.h.

Description

Socket API module calls this function in repeatedly when calls each APIs (connect()、accept()、send()、sendto()、recv()、recvfrom()) as blocking mode. And, socket API module calls this function in repeatedly when calls closesocket() as blocking/non-blocking mode.
Please call system call that can switch the task (in case ITRON, dly_tsk()) when use the Realtime OS.
Please do not call the function when not use the Realtime OS.

Example

```
void r_socket_task_switch(int sock)
{
#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()

Waiting for select() process.

Format

```
void r_socket_task_switch_select(void)
```

Parameters

None.

Return Values

None.

Properties

Prototyped in r_socket_rx_if.h.

Description

Socket API module calls this function in repeatedly when calls select().
Please call system call that can switch the task (in case ITRON, dly_tsk()) when use the Realtime OS.
Please do not call the function when not use the Realtime OS.

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()

Semaphore lock function.

Format

```
int r_socket_sem_lock(void)
```

Parameters

None.

Return Values

SOCKET_ERROR	*Operation failed*
E_OK	*Operation successful*

Properties

Prototyped in r_socket_rx_if.h.

Description

This function is called when SOCKET_IF_USE_SEMP=1.

Please call the semaphore lock function when you use the Realtime OS.

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()

Semaphore release function.

Format

```
int r_socket_sem_release(void)
```

Parameters

None.

Return Values

SOCKET_ERROR	*Operation failed*
E_OK	*Operation successful*

Properties

Prototyped in r_socket_rx_if.h.

Description

This function is called when SOCKET_IF_USE_SEMP=1.

Please call the semaphore release function when you use the Realtime OS.

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. Note

5.1 Several Ethernet channel support.

This module support only 1 port.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.32	Feb 01, 2019	-	Changes associated with functions: Added support setting function of configuration option Using GUI on Smart Configurator. [Description] Added a setting file to support configuration option setting function by GUI.
1.31	Oct 01, 2016	-	Updated the xml file for FIT. Deleted USE_BSD_NON_BLOCKING macro. Added SOCKET_TCP_WINSIZE macro. Changed R_SOCKET_Init() API name to R_SOCKET_Open(). Added R_SOCKET_Close(). Omitted Ether-2 channels support. Added section4 and section5 in this document.
1.30	Sep 15, 2015	-	Changed: added fcntl(), select() and errno for each API. Update descriptions of send/sendto/accept API.
1.22	Feb 12, 2015	-	Fixed source code.
1.21	Jun 31, 2015	-	Changed FIT Module name. Added Support MCUs.
1.20	Jun 31, 2015	-	Changed: Support for new T4 that can handle 2 ETHER channels.
1.10	Apr 01, 2014	-	Changed: Revision number corresponds to the software version.
1.00	-	-	First edition issued

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.2