User's Manual

RZ/T1 Group

µNet3/BSD User's Manual

・RZ/T1

All information of mention is things at the time of this document publication, and Renesas Electronics may change the product or specifications that are listed in this document without a notice. Please confirm the latest information such as shown by website of Renesas

Document number : R01US0204EJ0200
Issue date : Nov 1, 2020
Renesas Electronics
www.renesas.com

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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate 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. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be 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, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.
(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Instructions for the use of product

In this section, the precautions are described for over whole of CMOS device.
Please refer to this manual about individual precaution.
When there is a mention unlike the text of this manual, a mention of the text takes first priority

1. Handling of Unused Pins

Handle unused pins in accord 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 LSI, associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

Access to reserved addresses is prohibited.

- The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

How to Use This Manual

1. Objective and Target Users

This manual was written to explain the functions and the usage of the BSD interface library to the target users, i.e. those who will be using this library software in the design of application systems. Target users are expected to understand the fundamentals of the programming language and microcomputers.

When using this software, take all points to note into account. Points to note are given in their contexts and at the final part of each section, and in the section giving usage notes.

The list of revisions is a summary of major points of revision or addition for earlier versions. It does not cover all revised items. For details on the revised points, see the actual locations in the manual.

Contents

1. Introduction

The µNet3/BSD socket API provides a BSD interface for running BSD applications on the µNet3 TCP/IP stack. The stack and API allow seamless operation of socket applications from the Linux or BSD environments.

This document describes how to use the µNet3/BSD API and restrictions related to the product.

2. Specification

2.1 Position in the POSIX Specification

The µNet3/BSD socket API is equivalent to 4.4 BSD-Lite. See Section 4, Supported API for the APIs supported in this document. Using the µNet3/BSD API allows applications to use both BSD-based socket APIs and µNet3-based APIs.

2.2 Differences from the µNet3

The µNet3/BSD API provides the following functionality for existing µNet3 in addition to a POSIX-compliant socket API.

- Multiple calls of socket API functions
- A select() function
- Loopback address
- Multicast grouping by sockets
- Listen queue of TCP sockets
- Socket errors

2.3 Compatibility of Symbol Name

The API functions, structures, and macros provided in the µNet3/BSD API are given the unique prefix “unet3_” to avoid conflicts between symbols in the compiler environment.

The POSIX standard symbol names used in applications are replaced by those ones with the prefix unique to µNet3/BSD by including sys/socket.h. This allows the operation of applications using BSD sockets under µNet3/BSD without changing the files of source code.

In this document, the symbols are indicated in the POSIX standard notation for readability.

3. Module Structure

3.1 Module Structure

Figure 3.1 shows the module blocks composing the µNet3/BSD.

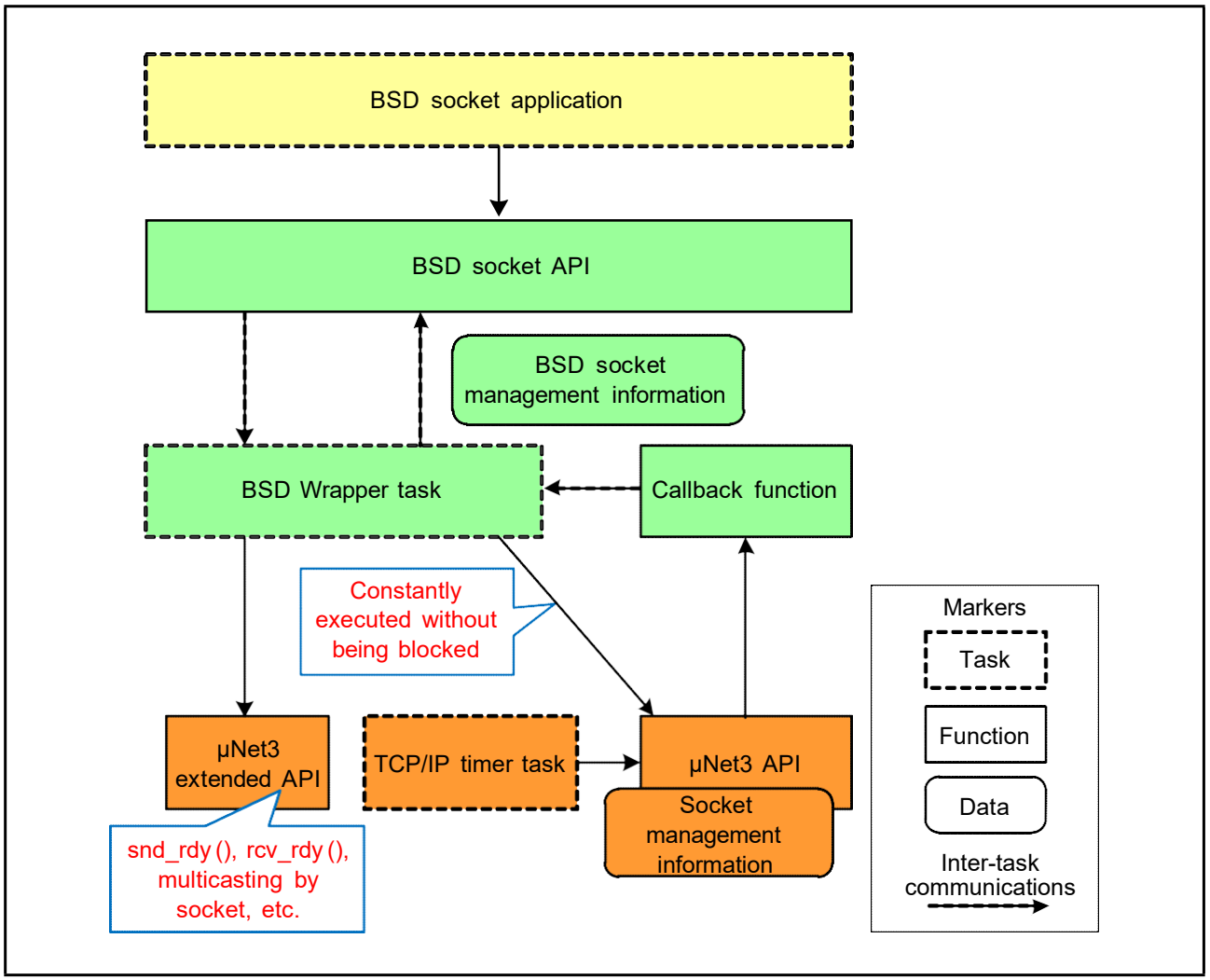

Figure 3.1

3.2 Header Structure

In the µNet3/BSD API, the POSIX-compliant header files are regarded as dummy files. These files include the open header file unet3_socket.h, an original file for µNet3/BSD. Table 3.1 lists the header files provided for the µNet3/BSD API.

Table 3.1 List of Header Files

Header File Name	Major Application
POSIX-Compliant Header Files (for Sockets)	
arpa/inet.h	Define the values for handling IP addresses
netinet/in.h	The address families AF_INET and AF_INET6 which include IP addresses and TCP/UDP port numbers. It is widely used on the internet.
netinet/ip.h	Define the IP-level options and IP packets.
netinet/tcp.h	Define the TCP-level options and TCP packets.
sys/socket.h	This contains declarations of the core functions for the BSD sockets and their data structures.
net/if.h	Interface related definitions
POSIX-Compliant Header Files (for Systems)	
sys/errno.h	Definitions of error codes
sys/ioctl.h	ioctl related definitions
sys/select.h	Definitions of functions including select and fd_set
sys/time.h	Definitions of functions including timeval type
sys/times.h	Definitions of functions including timeval type
sys/unistd.h	Standard header related to the UNIX standard
µNet3/BSD Original Header Files	
unet3_cfg.h	User-configuration definitions
unet3_socket.h	Open header file which defines the socket APIs
unet3_sys.h	Header file which defines types and macros specific to the BSD platform. The header files to be included for system integration of BSD applications are collected in this file.
unet3_wrap.h	For internal control
bsd_param.h	For internal control

3.3 Source Files

Source files used in the µNet3/BSD API are shown below.

When used in an application, *.c programs under the bsd folder should be incorporated.

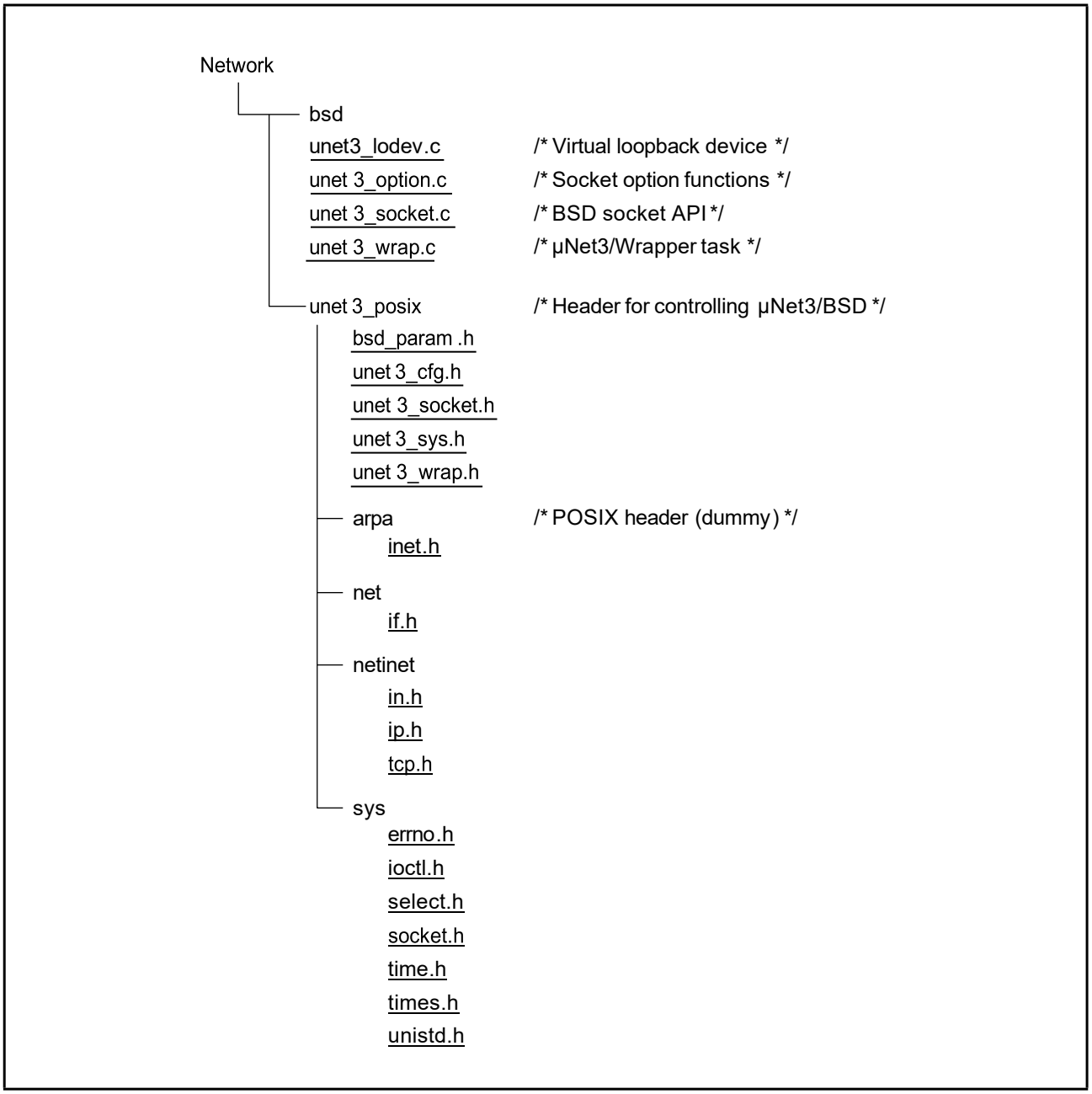

Figure 3.2

4. Supported API

4.1 Supported API Functions

Table 4.1 lists the API functions provided by the µNet3/BSD.

Table 4.1 List of API Functions

API Function	Description	Header for Inclusion
unet3_bsd_init	Initialize the µNet3/BSD	“sys/socket.h”
get_errno	Get the errnos for individual tasks	“sys/errno.h”
socket	Create an endpoint for communication	“sys/socket.h”
bind	Assign a name to a socket	“sys/socket.h”
listen	Waits for a connection on a socket	“sys/socket.h”
accept	Accept a connection on a socket	“sys/socket.h”
connect	Make a connection on a socket	“sys/socket.h”
send	Transmit a message to a socket	“sys/socket.h”
sendto	Transmit a message to a socket	“sys/socket.h”
recv	Receive a message from a socket	“sys/socket.h”
recvfrom	Receive a message from a socket	“sys/socket.h”
shutdown	Cause parts of a full-duplex connection on the socket to be shut down	“sys/socket.h”
close	Close a file descriptor (socket)	“sys/unistd.h”
select	Synchronous I/O multiplexing	“sys/select.h”
getsockname	Retrieve the name of a socket	“sys/socket.h”
getpeername	Retrieve the name of the peer connected to a socket	“sys/socket.h”
getsockopt	Retrieve options associated with a socket	“sys/socket.h”
setsockopt	Manipulate options associated with a socket	“sys/socket.h”
ioctl	Control hardware devices (sockets)	“sys/ioctl.h”
inet_addr	Internet address handling routine	“arpa/inet.h”
inet_aton	Internet address handling routine	“arpa/inet.h”
inet_ntoa	Internet address handling routine	“arpa/inet.h”
if_nametoindex	Map a network interface name to its corresponding index	“net/if.h”
if_indextoname	Map an interface index to its corresponding name	“net/if.h”
rresvport	Acquire a socket with a port bound to it	“sys/unistd.h”
getifaddrs	Retrieve the address of the interface	“sys/types.h”
freeifaddrs	Free the address of the interface	“sys/types.h”

4.2 Detail for Individual API Functions

socket (create an endpoint for communication)

Format

```
#include "sys/socket.h"
int socket(int domain, int type, int protocol);
```

Parameters			
	int	domain	Domain
	int	type	Communication type
	int	protocol	Protocol

Returned value		
	int	Created socket FD. This function returns -1 on occurrence of an error.

errno		
	ENOMEM	The number of sockets that can be created has been exceeded.
		Message buffer has been completely used up.
	EINVAL	An invalid parameter was specified.
	EINTR	Wait state was forcibly released.

- Allowed domains are AF_INET and AF_INET6 only.
- Allowed communication types are SOCK_STREAM and SOCK_DGRAM only.
- Set any value for the protocol as it is not used in this function.
- The number of sockets that can be created at the same time (the sum of TCP and UDP) is the value defined by #define CFG_NET_SOC_MAX.
- The number of TCP sockets that can be created at the same time is the value defined by #define CFG_NET_TCP_MAX.
- Setting 0 as the local port of a socket is not allowed. A socket is assigned a temporary local port number immediately after it is created.

bind (assign a name to a socket)

Format

```
#include "sys/socket.h"
int bind(int sockfd, const struct sockaddr *addr, unsigned int addrlen);
```

Parameters			
	int	sockfd	File descriptor of the socket
	const struct sockaddr *	addr	Local address
	unsigned int	addrlen	Local address length
Returned value			
	int	Result of processing. This function returns 0 on success and -1 if an error occurred.	
errno			
	EINVAL	An invalid parameter was specified.	
	ENOMEM	Message buffer has been completely used up.	
	EBADF	Invalid socket FD for binding.	
	EPIPE	Invalid socket object.	
	EAFNOSUPPORT	Unsupported address family.	
	EADDRINUSE	Address already in use.	
	EADDRNOTAVAIL	Cannot assign requested address.	
	EINTR	Wait state was forcibly released.	

- Local address should be set with the type struct sockaddr_in.
- The only allowed addresses as the IP address (IPv4) for the local address are the one set for the device or INADDR_ANY(unspecified).
- If the user sets PORT_ANY(0) as the port number of the local address, a port number is assigned by the protocol stack.
- The only allowed local address length is sizeof(struct sockaddr_in) (= 16).
- Set any value for the “sin_len”, a member of the type struct sockaddr_in, as it is not used in this function.
- To start reception, including listening for incoming connections from TCP (listen()) and receiving UDP packets (recv(), recvfrom()), the user needs to specify the target socket and execute the bind() function in advance.
- Binding to the well-known port numbers (1 to 1023) is also allowed.

listen (waits for a connection on a socket)

Format

```
#include "sys/socket.h"
int listen(int sockfd, int backlog);
```

Parameters			
	int	sockfd	File descriptor of the socket
	int	backlog	Backlog
Returned value			
	int	Result of processing. This function returns 0 on success and -1 if an error occurred.	
errno			
	EINVAL	An invalid parameter was specified, or the file descriptor of the TCP socket that has already been connected.	
	ENOMEM	Message buffer has been completely used up.	
	EBADF	Invalid socket FD for listening to.	
	EPROTONOSUPPORT	Unsupported protocol (non-TCP socket).	
	EINTR	Wait state was forcibly released.	

- This function makes the TCP socket listen for an incoming connection.
- Allowed file descriptors are those ones for TCP sockets.
- The maximum number of back logs is defined by #define CFG_NET_TCP_MAX -1.

accept (accept a connection on a socket)

Format

```
#include "sys/socket.h"
int accept(int sockfd, struct sockaddr *addr, unsigned int *addrlen);
```

Parameters		
int	sockfd	File descriptor of the socket
struct sockaddr *	addr	Remote address (output)
unsigned int *	addrlen	Remote address length (output)

Returned value	
int	The connected socket FD. This function returns -1 if an error occurred.

errno	
EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	The program is not listening to the specified socket.
EAGAIN	No connections have been made (in asynchronous network).
ETIMEDOUT	Connection attempt timed out (when a timeout is set).
EINTR	Wait state was forcibly released.

- Allowed file descriptors are those ones for TCP sockets for which the listen() function succeeded.
- The remote address is set with the type struct sockaddr_in*.
- If no connections were established, this function blocks further processing until an attempt of connection from a remote party.

connect (make a connection on a socket)

connect (make a connection on a socket)

Format

```
#include "sys/socket.h"
int connect(int sockfd, const struct sockaddr *addr, unsigned int addrlen);
```

Parameters			
	int	sockfd	File descriptor of the socket
	const struct sockaddr *	addr	Remote address
	unsigned int	addrlen	Remote address length

Returned value		
	int	Result of processing. This function returns 0 on success and -1 if an error occurred.

errno		
	EINVAL	An invalid parameter was specified.
	ENOMEM	Message buffer has been completely used up.
	EBADF	An invalid socket FD to connect.
	EHOSTUNREACH	Connection attempted to an inaccessible node.
	ECONNREFUSED	Connection refused by server.
	EAFNOSUPPORT	Unsupported address family.
	EISCONN	The socket is already connected.
		Listening to the socket is currently in progress.
	EALREADY	A connection request is already in progress.
	EAGAIN	A connection request is in progress (in asynchronous network).
	ETIMEDOUT	Connection attempt timed out (when a timeout is set).
	EINTR	Wait state was forcibly released.

- The function connect() operates and behaves differently depending on the protocol of the specified socket FD and the type of transfer.
- When connecting to a TCP socket, the µNet3/BSD API transmits the SYN signal to the address of the remote target and attempts connection to it. This only applies to TCP sockets other than those which are currently connected or for which waiting for a connection is in progress.
- In transmission through a UDP socket, the address of the remote target is regarded as the destination of transmission. If an address different from that of the remote target is set in the sendto() function, the given address is regarded as that of the destination for transmission.
- Setting AF_UNSPEC in the sa_family member of the remote address clears the setting mentioned above.
- The µNet3/BSD API differs from the POSIX specification in that it does not apply filtering of remote addresses in reception through UDP sockets.
- The µNet3/BSD API differs from the POSIX specification in that it cannot reissue a connection request for a TCP socket whose input and output are driven asynchronously, once a connection has been established by the connect() function. For example, if the ability to write to the target TCP socket has been ensured by select() after EAGAIN was returned in response to connect(), the session with the socket has been established, so the transmission and reception of data are possible.
- After a connect() function has been issued to a TCP socket set as asynchronous and processing in response is completed, if the behavior is as expected in response to a next connect() function to be issued; the return value is 0 on successful connection and the error number set in the previous processing is returned as errno. In case of failure to connect, the return value is -1 and the error number corresponding to the source of the error is returned as errno. Issuing a further connect() function is required for another connection request because exit from the processing is without a handshake regardless of whether connection failed or was successful.

send (transmit a message to a socket)

Format

```
#include "sys/socket.h"
int send(int sockfd, const void *buf, unsigned int len, int flags);
```

Parameters		
int	sockfd	File descriptor of the socket
const void *	buf	Source address of the data for transmission
unsigned int	len	Length of the data for transmission
int	flags	Flag

Returned value	
int	Number of bytes of the transmitted data. This function returns -1 if an error occurred.

errno	
EINVAL	An invalid parameter was specified (no address assigned to buf).
ENOMEM	Message buffer has been completely used up.
	The size of the network buffer does not match the value set in len, or the value in len is 0.
EBADF	An invalid socket FD for sending.
EPIPE	Invalid socket object.
EDESTADDRREQ	Destination address is not specified (UDP socket).
ENOTCONN	The socket is not connected (TCP socket).
EACCES	The attempted transmission was blocked because broadcast transmission is not allowed.
EAGAIN	Transmission is in progress (in asynchronous network).
ETIMEDOUT	Connection attempt timed out (when a timeout is set).
EINTR	Wait state was forcibly released.

- Valid values for the length of the data for transmission are between 1 and 65535.
- The behavior differs from that in the POSIX specification in that the transmission of 0-byte UDP packets is not allowed.
- If the MSG_DONTWAIT flag has been set in transmission to a UDP socket, the transmission is deemed successful at the point when the data have been placed in a queue in a lower layer*1.

Note 1. "Lower layer" above refers to processing for address resolution in the IP layer or for asynchronous transmission in the link layer.

sendto (transmit a message to a socket)

Format

```
#include "sys/socket.h"
int sendto(int sockfd, const void *buf, unsigned int len, int flags, const struct sockaddr
*dest addr, unsigned int addrlen);
```

Parameters		
int	sockfd	File descriptor of the socket
const void *	buf	Source address of the data for transmission
unsigned int	len	Length of the data for transmission
int	flags	Flag
const struct sockaddr *	dest_addr	Address of the destination for transmission
unsigned int	addrlen	Size of the address of the destination

Returned value	
int	Number of the bytes of the transmitted data. This function returns -1 if an error occurred.

errno	
EINVAL	An invalid parameter was specified (no address assigned to buf).
ENOMEM	Message buffer has been completely used up.
	The size of the network buffer does not match the value set in len, or the value in len is 0.
EBADF	An invalid socket FD for the sendto operation.
EPIPE	Invalid socket FD.
EDESTADDRREQ	Destination address is not specified (UDP socket).
ENOTCONN	The socket is not connected (TCP socket).
EACCES	The attempted transmission was blocked because broadcast transmission is not allowed.
EAGAIN	Transmission is in progress (in asynchronous network).
ETIMEDOUT	Connection attempt timed out (when a timeout is set).
EINTR	Wait state was forcibly released.

- Valid values for the length of the data for transmission are between 1 and 65535.
- The parameters for the address of the destination for transmission and the size of the address of the destination are not used when connecting to TCP sockets.
- The behavior differs from that in the POSIX specification in that the transmission of 0-byte UDP packets is not allowed.
- If the MSG_DONTWAIT flag has been set in transmission to a UDP socket, the transmission is deemed successful at the point when the data have been placed in a queue in a lower layer*1.

Note 1. "Lower layer" above refers to processing for address resolution in the IP layer or for asynchronous transmission in the link layer.

recv (receive a message from a socket)

Format

```
#include "sys/socket.h"
int recv(int sockfd, void *buf, unsigned int len, int flags);
```

Parameters		
int	sockfd	File descriptor of the socket
void *	buf	Address of the reception buffer
unsigned int	len	Length of the reception buffer
int	flags	Flag

Returned value	
int	Number of the bytes of the received data including 0 byte. This function returns -1 if an error occurred.

errno	
EINVAL	An invalid parameter was specified (no address assigned to buf).
ENOMEM	Message buffer has been completely used up.
	The size of the network buffer does not match the value set in len, or the value in len is 0.
EBADF	An invalid socket FD for the recv operation.
EPIPE	An invalid socket FD was specified.
ENOTCONN	The socket is not connected (TCP socket).
EAGAIN	No packet has been received (in asynchronous network).
ETIMEDOUT	Connection attempt timed out (when a timeout is set).
EINTR	Wait state was forcibly released.

- Setting the MSG_PEEK flag causes the receive operation to return packet of the receive queue without removing that data from the queue. Thus, a subsequent receive call will return the same packet.
- Valid values for the length of the reception data are between 1 and 65535.
- If no packets are received, this function blocks further processing until packet reception.
- This function returns an error if connection with a remote party has not been established.
- This function returns 0 if the TCP socket is disconnected from the remote party.

recvfrom (receive a message from a socket)

Format

```
#include "sys/socket.h"
int recvfrom(int sockfd, void *buf, unsigned int len, int flags, struct sockaddr *src_addr,
unsigned int *addrlen);
```

Parameters		
int	sockfd	File descriptor of the socket
void *	buf	Reception buffer address
unsigned int	len	Reception buffer length
int	flags	Flag
struct sockaddr *	src_addr	Source address of the data for transmission
unsigned int *	addrlen	Size of the source address

Returned value	
int	Number of the bytes of the received data including 0 byte. This function returns -1 if an error occurred.

errno	
EINVAL	An invalid parameter was specified (no address assigned to buf).
ENOMEM	Message buffer has been completely used up.
	The size of the network buffer does not match the value set in len, or the value in len is 0.
EBADF	An invalid socket FD for the recvfrom operation.
EPIPE	An invalid socket FD was specified.
ENOTCONN	The socket is not connected (TCP socket).
EAGAIN	No packet has been received (in asynchronous network).
ETIMEDOUT	Connection attempt timed out (when a timeout is set).
EINTR	Wait state was forcibly released.

- Setting the MSG_PEEK flag causes the receive operation to return packet of the receive queue without removing that data from the queue. Thus, a subsequent receive call will return the same packet.
- Valid values for the length of the reception data are between 1 and 65535.
- If no packets are received, this function blocks further processing until packet reception.
- This function returns an error if connection with a remote party has not been established.
- This function returns 0 if the TCP socket is disconnected from the remote party.
- The parameters for the source address of the data for transmission and the size of the source address are not used when connecting to TCP sockets.

shutdown (cause parts of a full-duplex connection on the socket to be shut down)

Format

```
#include "sys/socket.h"
int shutdown(int sockfd, int how);
```

Parameters			
	int	sockfd	File descriptor of the socket.
	int	how	Type of shut down
Returned value			
	int	Result of processing. This function returns 0 on success and -1 if an error occurred.	
errno			
	EINVAL	An invalid parameter was specified.	
	ENOMEM	Message buffer has been completely used up.	
	EBADF	An invalid socket FD to shutdown.	
	EPIPE	The socket is not connected (TCP socket).	
	EINTR	Wait state was forcibly released.	

- The only allowed types of shutdown are SHUT_WR and SHUT_RDWR.

close (close a socket)

Format

```
#include "sys/unistd.h"
int close(int fd);
```

Parameters		
int	fd	File descriptor of the socket
Returned value		
int	Result of processing. This function returns 0 on success and -1 if an error occurred.	
errno		
EINVAL	An invalid parameter was specified.	
ENOMEM	Message buffer has been completely used up.	
EBADF	An invalid socket FD to close.	
EINTR	Wait state was forcibly released.	

- If a TCP session is active when this function is called, the socket will be closed after the session has been cut off.
- Once the socket with the given FD is closed, it cannot be used again until a new connection is established.

select (synchronous I/O multiplexing)

Format

```
#include "sys/select.h"
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval
*timeout);
```

Parameters		
int	nfds	An integer one greater than the highest file descriptor in readfds and writefds. When adding file descriptors to either of the sets, increment this value by one.
fd_set *	readfds	A set of file descriptors to be checked for readability.
fd_set *	writefds	A set of file descriptors to be checked for writability.
fd_set *	exceptfds	A set of file descriptors to be checked for exceptional conditions (not supported).
struct timeval *	timeout	Time until expiration of the monitoring period

Returned value	
int	The total number of file descriptors to be checked for readability or writability. This function returns 0 on timeout and -1 if an error occurred.

errno	
EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	The socket with the specified FD does not support select().

- The argument exceptfds is not used in this function.
- The µNet3/BSD API differs from the POSIX specification in that, when this function is executed for a socket file descriptor immediately after it has been created, the function allows writing but not reading if it is for a UDP socket (reading is also possible if a packet has already been received). The function allows reading (but not writing) if it is for a TCP socket.

getsockname (retrieve the name of a socket)

Format

```
#include "sys/socket.h"
int getsockname(int sockfd, struct sockaddr *addr, unsigned int *addrlen);
```

Parameters		
int	sockfd	File descriptor of the socket.
struct sockaddr *	addr	Pointer to the buffer where the socket address is stored.
unsigned int *	addrlen	Size of the buffer where the socket address is stored.
Returned value		
int	Result of processing. This function returns 0 on success and -1 if an error occurred.	
errno		
EINVAL	An invalid parameter was specified.	
ENOMEM	Message buffer has been completely used up.	
EBADF	An invalid socket FD for the getsockname operation.	
EINTR	Wait state was forcibly released.	

- The value in *addrlen should be the size of sockaddr_in (16 bytes or more).
- The address is bound to a socket when the following API functions are called.

  bind()

  connect()

  accept()

  send/sendto()

  recv/recvfrom()

  If a function from the above list fails, the value of the address associated with the socket will be undefined.

getpeername (retrieve the name of the peer connected to a socket)

Format

```
#include "sys/socket.h"
int getpeername(int sockfd, struct sockaddr *addr, unsigned int *addrlen);
```

Parameters		
int	sockfd	File descriptor of the socket.
struct sockaddr *	addr	Pointer to the buffer where the remote address is stored.
unsigned int *	addrlen	Size of the buffer where the remote address is stored.
Returned value		
int	Result of processing. This function returns 0 on success and -1 if an error occurred.	
errno		
EINVAL	An invalid parameter was specified.	
ENOMEM	Message buffer has been completely used up.	
EBADF	An invalid socket FD for the getpeername operation.	
ENOTCONN	Destination address required.	
EINTR	Wait state was forcibly released.	

- The value in *addrlen should be the size of sockaddr_in (16 bytes or more).
- For a TCP connection, this function only allows retrieval of the address of the remote party to which the TCP socket is connected.
- For a UDP connection, this function only allows retrieval of the address of a remote party with an address previously specified in a connect or sendto function, or of a socket which has received packets.

getsockopt (retrieve options associated with a socket)

Format

```
#include "sys/socket.h"
int getsockopt(int sockfd, int level, int optname, void *optval, unsigned int *optlen);
```

Parameters		
int	sockfd	File descriptor of the socket.
int	level	The level of the option
int	optname	Option name
void *	optval	A pointer to the buffer where the value of the option is to be stored.
unsigned int *	optlen	The size of the buffer pointed to by optval.

Returned value	
int	Result of processing. This function returns 0 on success and -1 if an error occurred.

errno	
EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	An invalid socket FD for the getsockopt operation.
EPROTONOSUPPORT	The option is not supported.
EINTR	Wait state was forcibly released.

setsockopt (manipulate options associated with a socket)

Format

```
#include "sys/socket.h"
int setsockopt(int sockfd, int level, int optname, const void *optval, unsigned int optlen);
```

Parameters		
int	sockfd	File descriptor of the socket
int	level	The level of the option
int	optname	Option name
const void *	optval	The buffer in which the values of the requested options are to be returned.
unsigned int	optlen	Size of the buffer pointed to by optival

Returned value	
int	Result of processing. This function returns 0 on success and -1 if an error occurred.

errno	
EINVAL	An invalid parameter was specified.
ENOMEM	Message buffer has been completely used up.
EBADF	An invalid socket FD for the setsockopt operation.
EPIPE	An invalid socket FD was specified.
EPROTONOSUPPORT	The option is not supported.
EINTR	Wait state was forcibly released.

- Allowed option levels are SOL_SOCKET, IPPROTO_IP and IPPROTO_TCP.
- The available option names at each option level are listed in the Section 5.1, List of Options.

- Allowed option levels are SOL_SOCKET, IPPROTO_IP and IPPROTO_TCP.
- The available option names at each option level are listed in the Section 5.1, List of Options.

ioctl (control hardware devices (sockets))

Format

```
#include "sys/ioctl.h"
int ioctl(int d, int request, ...);
```

Parameters			
	int	d	File descriptor of the socket
	int	request	Request
	...		Parameter for the request

Returned value		
	int	Result of processing. This function returns 0 on success and -1 if an error occurred.

errno		
	EINVAL	An invalid parameter was specified.
	EBADF	An invalid socket FD was specified.
	ENOMEM	Message buffer has been completely used up.
	EFAULT	The parameter for the request is not usable.
	EINTR	Wait state was forcibly released.

- Allowed settings for the request parameter are listed in the table below.

Request Code	Meaning	Parameter
FIONBIO	Enable non-blocking communications	1 (enabling) or 0 (disabling)
FIONREAD	Get the number of bytes waiting to be read on a socket.	(unsigned int *) &nread

- See Section 6.1, Non-Blocking Mode for details on non-blocking mode.
- The value obtained by the option FIONREAD is the size of a whole packet in the reception window buffer in the case of a TCP socket or the size of a received packet block (header only) to be received in the next transaction in the case of a UDP socket.

inet_addr (Internet address handling routine)

Format

```
#include "arpa/inet.h"
unsigned int inet addr(const char *cp);
```

Parameters			
	const char *	cp	The IP address in dot-notation
Returned value			
	unsigned int	The IP address converted into binary data in network byte order	
errno			
	Not specified		

- The function returns 0 if conversion failed.

inet_aton (Internet address handling routine)

Format

```
#include "arpa/inet.h"
int inet aton(const char *cp, struct in addr *inp);
```

Parameters		
const char *	cp	The IP address in dot-notation
struct in_addr *	inp	Pointer to the buffer where the post-conversion binary value of the IP address is stored in network byte order.

Returned value	
int	Result of processing. This function returns 0 on success and -1 if an error occurred.

errno
Not specified

- The function returns -1 if conversion failed.

inet_ntoa (Internet address handling routine)

Format

```
#include "arpa/inet.h"
char *inet ntoa(struct in addr in);
```

Parameters		
struct in_addr	in	IP address as binary data in network byte order

Returned value	
char *	The IP address converted into dot-notation

errno
Not specified

- The string is returned in a statically allocated buffer in the area for the IP address converted into dot-notation, and will be overwritten by subsequent calls.

if_nametoindex (map a network interface name to its corresponding index)

Format

```
#include "net/if.h"
unsigned int if nametoindex(const char *ifname)
```

Parameters		
const char *	ifname	Interface name

Returned value	
unsigned int	The index of the interface. This function returns 0 if an error occurred.

errno	
ENXIO	An interface with the given name does not exist.

- The setting for the name of the interface is based on the device name in µNet3 (gNET_DEV[index-1].name[8]).

if_indextoname (map an interface index to its corresponding name)

Format

```
#include "net/if.h"
char *if_indextoname(unsigned int ifindex, char *ifname)
```

Parameters			
	unsigned int	ifindex	Interface index
	char *	ifname	Pointer to the buffer where the interface name is stored.
Returned value			
	char*	Result of processing. This function returns ifname on success and null if an error occurred.	
errno			
	ENXIO	No index found for the interface.	

- The setting for the name of the interface is based on the device name in µNet3 (gNET_DEV[index-1].name[8]).

rresvport (acquire a socket with a port bound to it)

Format

```
#include "sys/unistd.h"
int rresvport(int *port)
```

Parameters			
	int*	port	Pointer to the buffer where the port number is stored.
Returned value			
	int	A socket file descriptor bound to a port. This function returns -1 if no socket is present.	
errno			
	Not specified		

getifaddrs (retrieve the address of the interface)

Format

```
#include "sys/types.h"
int getifaddrs(struct ifaddrs **ifap)
```

Parameters		
struct ifaddrs**	ifap	The address of the first item in the list of network interfaces

Returned value	
int	Result of processing. This function returns 0 on success and -1 if an error occurred.

errno	
ENOMEM	Failure to acquire the area where the information about the interfaces is stored.

- This function acquires the information about the interfaces in the chain for the devices (CFG_DEV_MAX) set in the application.
- On success, this function stores the following values in the argument ifap.
  (*ifap)->ifa_next: a pointer to the next structure in the list, or null if this is the last item in the list
  (*ifap)->name: a pointer to the interface name
  (*ifap)->ifa_flags: the device number
  (*ifap)->ifa_addr: a pointer to the sockaddr structure which contains the IP address of the interface.
  (*ifap)->ifa_netmask: a pointer to the sockaddr structure which contains the subnet mask.
  (*ifap)->ifa_ifu and (*ifap)->ifa_data: not used in this function.
- The data returned by getifaddrs() is dynamically allocated and should be freed by using freeifaddrs() after the function succeeds.

freeifaddrs (free list of interface information)

Format

```
#include "sys/unistd.h"
void freeifaddrs(struct ifaddrs *ifap)
```

Parameters		
struct ifaddrs*	ifap	The address of the first item in the list of network interfaces

Returned value
void

errno
Not specified

- This function frees the list of interface information acquired by getifaddrs().

5. Socket Options

5.1 List of Options

Table 5.1, List of Options is the options acquired or set by using the functions setsockopt() and getsockopt(). If a value other than those listed below is specified, the function returns -1. In the list, “GET” represents operations to which getsockopt() is applicable and “SET” represents operations to which setsockopt() is applicable.

Table 5.1 List of Options

Option Name	Type	Description
SOL_SOCKET Level		
SO_ACCEPTCONN	int	Retrieve the state of a TCP socket, whether it is in listening mode or not. Only GET is applicable.
SO_BROADCAST	int	Configure a socket for transmitting UDP broadcast data. Both GET and SET are applicable.
SO_DOMAIN	int	Acquire the socket domain. Only GET is applicable.
SO_ERROR	int	Acquire a socket error. Only GET is applicable.
SO_KEEPALIVE*1	int	Enable sending of keepalive packets by the TCP socket. Only SET is applicable.
SO_RCVBUF	int	Make settings for the reception buffer. This is the number of bytes in reception windows for TCP and the number of received packets (queue size) for the UDP. Both GET and SET are applicable.
SO_RCVBUFFORCE	int	Same as SO_RCVBUF.
SO_RCVTIMEO	timeval	Specify the timeout value for a receiving socket. Both GET and SET are applicable.
SO_SNDTIMEO	timeval	Specifies the timeout value for a sending socket. Both GET and SET are applicable.
SO_TYPE	int	Retrieves the socket type. Only GET is applicable.
SO_REUSEADDR	int	Allows a socket to forcibly bind to a local port that is already in use by another socket. Both GET and SET are applicable.
IPPROTO_IP Level		
IP_ADD_MEMBERSHIP	ip_mreqn	Joins the multicast groups specified, applicable to UDP sockets only. Only SET is applicable.
IP_DROP_MEMBERSHIP	ip_mreqn	Drops membership of a multicast group. Only SET is applicable.
IP_MTU	int	Retrieve the path MTU. Only GET is applicable.
IP_MULTICAST_TTL	int	Set the TTL (time-to-live) for transmitted multicast packets. Both GET and SET are applicable.
IP_TOS	int	Set the TOS (type of service) for transmitted IP packets. Both GET and SET are applicable.
IP_TTL	int	Set the TTL for transmitted IP packets. Both GET and SET are applicable.
IPPROTO_TCP Level		
TCP_KEEPCNT*1	int	Specifies the number of keepalive probes for TCP sockets. Only SET is applicable.
TCP_KEEPIDLE*1	int	Specifies the interval of inactivity that causes the TCP to generate a keepalive transmission for an application that requests them. Only SET is applicable.
TCP_KEEPINTVL*1	int	Specifies the interval between keepalive probes for TCP sockets. Only SET is applicable.
TCP_MAXSEG	int	Specifies the MSS (maximum segment size) value for TCP packets. Both GET and SET are applicable.

Note 1. Enable or disable the TCP keepalive option (SO_KEEPALIVE) before making a TCP connection. Settings associated with the option keepalive, including this one, are shared among all sockets.

6. Capabilities

6.1 Non-Blocking Mode

The ioctl() function sets the API call for a socket in non-blocking mode (or blocking mode). All API functions are set to blocking mode as the initial value. There are some cases where an API in non-blocking mode sets EAGAIN as errno and returns -1. The APIs which operate in non-blocking mode and the conditions for returning EAGAIN as errno, and the expected behaviors of the application are listed in Table 6.1, Non-Blocking APIs.

Note that setting the timeout option for a socket is not effective for the APIs which behave in non-blocking mode. Furthermore, in µNet3/BSD, even if an API function is set to non-blocking mode, it may need to wait for the task to wake up after being called due to the specification for inter-task transfer.

Table 6.1 Non-Blocking APIs

API	Condition	Application Behavior
connect	If the target is a TCP socket, the returned value is always -1, and the error code otherwise is EAGAIN.	Even after -1 is returned, the TCP socket keeps sending SYN packets for the specified time while waiting for SYN and ACK packets from the remote party. The socket is monitored by the select function with the parameter writefds, for readiness for writing on reception of SYN and ACK. Once the socket becomes ready for writing, further execution of the connect function is not needed.
accept	If there is no connection attempted to the listen socket, the returned value is -1 and the error code is EAGAIN.	The socket is monitored by the select function with the parameter readfds, for readiness for reading on reception of SYN. Once the socket becomes ready for reading, the accept function is executed again.
send sendto	When the send buffer is full in the transfer with the TCP sockets and when a transmission is in progress in the transfer with the UDP sockets, the error code is EAGAIN.	EAGAIN for the functions send and sendto means that packet transmission failed (and will not be transmitted again) due to conditions of sockets.
recv recvfrom	When no packet has been received, the error code is EAGAIN.	The socket is monitored by the select function with the parameter readfds, for readiness for reading on reception of packets from the remote party. Once the socket becomes ready for reading, the recv function is executed again.

6.2 Loopback

When local loopback addresses (127.0.0.1 to 127.255.255.254) are specified as destination for transmission, the transmitted packets are conveyed to the network interface of the local device.

In µNet3/BSD, the loopback addresses are not assigned to any specific device interface and are regarded as send-only addresses. Therefore, they cannot be used in the bind() operation.

6.3 Error Processing

The symbol errno is the only global variable used in the µNet3/BSD API. Its value is updated on the occurrence of errors during the execution of API functions. When the user executes API functions from multiple tasks, we recommend acquiring the last errno by using get_errno(), to maintain consistency between errno values and the errors.

Format

```
#include "sys/errno.h"
int get errno(void)
```

Parameters		
	void	
Returned value		
	int	The errno of the last error to have occurred during the API function calls by a given task.
errno		
	Not specified	

- The errno for each task will be stored in the global variable UW tsk_errno[], provided in the application. The array should have the same number of elements as the maximum number of tasks.

6.4 List of errno

The values defined for errno may vary according to the compiler.

[Definition pattern 1]

Applicable compilers:

- RealView Developer Suite from Arm
- Embedded Workbench (EWARM) from IAR
- Code Composer Studio from TI
- GNU C Compiler

errno	Value	Description
EINTR	4	Wait state of the API was forcibly released
ENXIO	6	No interface found
EBADF	9	Invalid socket file descriptor
ENOMEM	12	Not enough memory
EACCES	13	Access for the requested process was blocked
EFAULT	14	Error in a parameter
ENODEV	19	Critical (or unknown) error in the system
EINVAL	22	Invalid parameter value
EPIPE	32	Invalid socket object
EAGAIN	35	Connection is blocked
EALREADY	37	The operation is already in progress
EDESTADDRREQ	39	Destination address required
EPROTONOSUPPORT	43	Protocol not supported
EAFNOSUPPORT	47	Address family not supported by protocol
EADDRINUSE	48	Address already in use
EADDRNOTAVAIL	49	Cannot assign requested address
EISCONN	56	The socket is already connected
ENOTCONN	57	The socket is not connected
ETIMEDOUT	60	Connection attempt timed out
ECONNREFUSED	61	Connection is refused by server
EHOSTUNREACH	65	Connection attempted for an inaccessible node

[Definition pattern 2]

Applicable compiler:

- CubeSuite+ * from Renesas Electronics

errno	Value	Description
ENXIO	*	No interface found
EBADF	*	Invalid socket file descriptor
ENOMEM	*	Not enough memory
EACCES	*	Access for the requested process was blocked
EFAULT	*	Error in a parameter
ENODEV	*	Critical (or unknown) error in the system
EINVAL	*	Invalid parameter value
EPIPE	*	Invalid socket object
EAGAIN	*	Connection is blocked
EALREADY	0x1025	The operation is already in progress
EDESTADDRREQ	0x1027	Destination address required
EPROTONOSUPPORT	0x102B	Protocol not supported
EAFNOSUPPORT	0x102F	Address family not supported by protocol
EADDRINUSE	0x1030	Address already in use
EADDRNOTAVAIL	0x1031	Cannot assign requested address
EISCONN	0x1038	The socket is already connected
ENOTCONN	0x1039	The socket is not connected
ETIMEDOUT	*	Connection attempt timed out
ECONNREFUSED	0x103D	Connection refused by server

- Restrictions
  - Do not use the name “errno” as the name of a variable where error codes are stored.
    The name “errno” is used as the variable defining errors in the CubeSuite+ standard library. With the µNet3/BSD API, use unet_errno instead.
  - Part of the values defined for errno are the same as those used in the compiler (as indicated with * (asterisk) in the above list)
    If the same name is defined for an errno value in the µNet3/BSD API and the CubeSuite+, the one in the compiler is used.

7. Implementing BSD Application

7.1 Source Code

An application which uses the µNet3/BSD API must be combined in projects with four files of source code from the Network/bsd/ folder (see Section 3.3, Source Files.)

Also, link the version of µNet3/BSD which is prepared for BSD (uNet3BSDxxxx.lib) as the library.

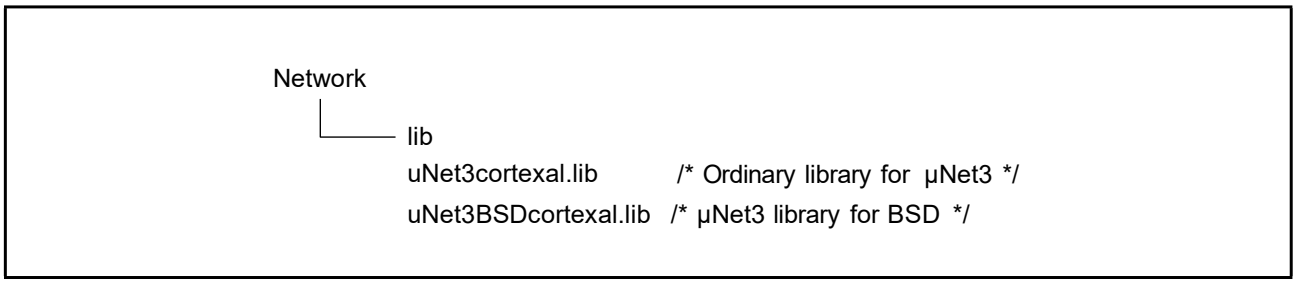

Figure 7.1

7.2 Include Path

An application which uses the µNet3/BSD API requires additional settings for “include” paths. The header file is found in the Network/bsd/unet3_posix folder with the POSIX-compatible files.

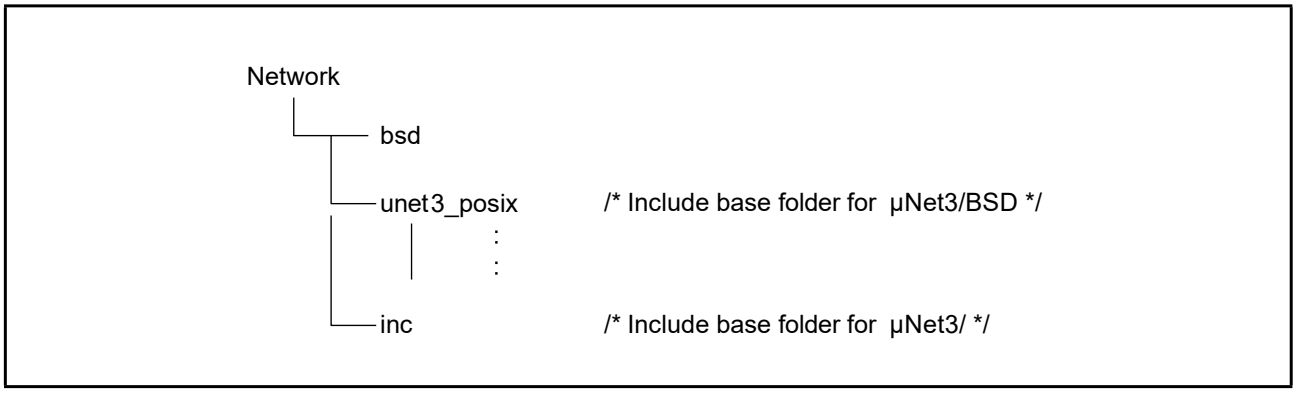

Figure 7.2

7.3 Configuration

In the µNet3/BSD API, the maximum number of sockets to be used in an application and tasks to be executed should be defined in the below macro in unet3_cfg.h in advance.

Maximum number of sockets
#define BSD_SOCKET_MAX

The maximum number of sockets, regardless of the protocol, shows the number of sockets that an application can create at the same time (including backlog from listening). This macro definition is used for defining the number of entries in the management table for BSD sockets and fd_set type settings, which will be described later. This value must be same as the maximum number of the sockets used by µNet3 (CFG_NET_SOC_MAX).

Number of application tasks
#define NUM_OF_TASK_ERRNO

The number of application tasks shows the number of tasks that can be created in the kernel. This macro definition is used for the number of entries in the management table for error codes, which will be described later. Set the number of tasks that can be created, regardless of whether you are using µNet3/BSD.

7.4 Defining Resources

Applications which use the µNet3/BSD API should provide resources required for operating the program, which are, the tables for managing information of the µNet3/BSD as listed below.

BSD socket management table
T_UNET3_BSD_SOC gNET_BSD_SOC[BSD_SOCKET_MAX];

This table defines a global variable as the number of elements BSD_SOCKET_MAX in the T_UNET3_BSD_SOC array.

Error code management table
UW tsk_errno[NUM_OF_TASK_ERRNO];

This table defines a global variable as the number of elements NUM_OF_TASK_ERRNO in the array of UW.

7.5 Kernel Objects

The Kernel objects used in the µNet3/BSD are shown below.

Resource name	Usage	ID
Task	BSD wrapper task	ID TSK_BSD_API
	Loopback device task	ID_LO_IF_TSK
Mailbox	Communication between BSD wrapper tasks	ID MBX_BSD_REQ
	Communication between loopback device tasks	ID_LO_IF_MBX
Memory pool	Message buffer	ID MPF_BSD_MSG

7.6 Initialization

When an application uses the socket API functions of the µNet3/BSD API, the module must be initialized in advance by calling the unet3_bsd_init() function. This operation should be performed after successful initialization of µNet3 and the device driver.

Format

```
#include "sys/socket.h"
ER unet3 bsd init(void)
```

Parameters	
void	
Returned value	
ER	Result of processing. This function returns E_OK on success and the error code if an error occurred.
Error code	
E_SYS	Initialization of the kernel object failed.

8. Appendix

8.1 Supported Compilers

The µNet3/BSD guarantees operation in the following compilers.

- RealView Developer Suite from Arm
- Embedded Workbench (EWARM) from IAR
- Code Composer Studio from TI
- GNU C Compiler
- CubeSuite+ from Renesas Electronics

Note: Restrictions are given depending on the compiler.

8.2 Sample Application

Sample applications using the µNet3/BSD API are included in the Sample folder. These sample programs are also available in the POSIX environment (Linux).

- API Console (sample_sockcmd.c)
  The user can run the API functions through the command prompt (with a UART connection) by input of the socket API function and required parameters. For details, refer to the Readme_command.txt.

8.3 Restrictions on Compilers

Restrictions are given to some compilers when they are used with µNet3/BSD.

- CubeSuite+ from Renesas Electronics
  – Do not use the name “errno” as the name of a variable where error codes are stored.
  The name “errno” is used as the variable defining errors in the CubeSuite+ standard library. With the µNet3/BSD API, use unet_errno instead.

Rev.	Date	Description	
		Page	Summary
1.00		—	First edition, issued
2.00	Nov 1, 2020	4. Supported API	
		11, 35, 36	Functions getifaddrs() and freeifaddrs(), added
		12 to 36	Additions and modifications to section 4.2, Detail for Individual API Functions
		19, 20	Option MSG_PEEK and related description, added
		28	Option FIONREAD and related description, added
		5. Socket Options	
		37	Complementary description of socket options regarding TCP Keep-Alive, added
		37	Operation of the option SO_BROADCAST and related description in section 5.1, List of Options, modified
		37	Option SO_REUSEADDR and related description in section 5.1, List of Options, added
		6. Capabilities	
		40	Description in section 6.4, List of errno, modified
		40, 41	Descriptions of EACCES and EHOSTUNREACH, added in section 6.4, List of errno
		8. Appendix	
		45	CubeSuite+, added to the supported compilers
		45	Section 8.3, Restrictions on Compilers, added

RZ/T1 Group User's Manual: µNet3/BSD

Publication Date: Rev.1.00 Jul. 27, 2013
Rev.2.00 Nov. 1, 2020

Published by: Renesas Electronics Corporation

RZ/T1 Group User's Manual

µNet3/BSD

Renesas Electronics Corporation

R01US0204EJ0200