User's Manual

R-IN32M4-CL3 Series

User's Manual: TCP/IP stack

- R-IN32M4-CL3

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 : R18UZ0079EJ0100
Issue date : Aug 31, 2021

Renesas Electronics

www.renesas.com

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. Purpose and target readers

This manual is intended for users who wish to understand the functions of Industrial Ethernet network LSI “R-IN32” for designing application of it.

It is assumed that the reader of this manual has general knowledge in the fields of electrical engineering, logic circuits, and microcontrollers.

Particular attention should be paid to the precautionary notes when using the manual. These notes occur within the body of the text, at the end of each section, and in the Usage Notes section.

The revision history summarizes the locations of revisions and additions. It does not list all revisions. Refer to the text of the manual for details.
The mark “<R>” means the updated point in this revision. The mark “<R>” let users search for the updated point in this document.

Related Documents	The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such. Please be understanding of this beforehand. In addition, because we make document at development, planning of each core, the related document may be the document for individual customers. Last four digits of document number(described as ****) indicate version information of each document. Please download the latest document from our web site and refer to it.

Documents related to R-IN32M4-CL3

Document Name	Document Number
R-IN32M4-CL3 User’s Manual	R18UZ0073EJ****
R-IN32M4-CL3 User’s Manual: Gigabit Ethernet PHY	R18UZ0075EJ****
R-IN32M4-CL3 User’s Manual: Board	R18UZ0074EJ****
R-IN32M4-CL3 Programming Manual: Driver	R18UZ0076EJ****
R-IN32M4-CL3 Programming Manual: OS	R18UZ0072EJ****
R-IN32M4-CL3 User’s Manual CC-Link IE Field	R18UZ0071EJ****
R-IN32M4-CL3 User's Manual TCP/IP stack	This Manual

2. Notation of Numbers and Symbols

Weight in data notation: Left is high-order column, right is low-order column

Active low notation:

xxxZ (capital letter Z after pin name or signal name)

or xxx_N (capital letter _N after pin name or signal name)

or xxnx (pin name or signal name contains small letter n)

Note:

explanation of (Note) in the text

Caution:

Item deserving extra attention

Remark:

Supplementary explanation to the text

Numeric notation:

Binary ··· xxxx , xxxxB or n'bxxxx (n bits)

Decimal ··· xxxx

Hexadecimal ··· xxxxH or n'hxxxx (n bits)

Prefixes representing powers of 2 (address space, memory capacity):

K (kilo)··· $2^{10} = 1024$

M (mega)··· $2^{20} = 1024^2$

G (giga)··· $2^{30} = 1024^3$

Data Type:

Word ··· 32 bits

Halfword ··· 16 bits

Byte ··· 8 bits

Contents

R-IN32M4-CL3 Series

User's Manual: TCP/IP stack

R18UZ0079EJ0100
Rev.1.00
Aug 31, 2021

1. Overview

This document explains TCP/IP and UDP/IP protocol stacks for R-IN32M4-CL3.

Function summary and Application Programming Interface (API) and application samples of TCP/IP protocol stack provided by Renesas are described in this document.

1.1 Features

TCP/IP stack is a compact stack optimized for R-IN32M4-CL3.

1.2 Key Functions

TCP/IP stack has key functions as follows:

- IPv4, ARP, ICMP, IGMPv2, UDP, TCP protocol
- DHCP client, DNS client, FTP server, HTTP server
- TCP/IP configuration
- TCP fast retransmit and fast recovery
- IP reconstructure and fragmentation
- Multiple network interface
- Source code for network applications
- Source code for network device drivers
- Protocol stack provided in library

1.3 Development environment

The development environment of TCP/IP protocol stack is described here.

1.3.1 Development tools

The sample software has been confirmed to work with the following tool chain.
This sample software adopts Arm® Cortex® Microcontroller Software Interface Standard (CMSIS)
Regarding the detailed information, please refer to the documentation of CMSIS.

Table 1.1 Development tools

LSI	Tool Chain	IDE	Compiler	Debugger	ICE
R-IN32M4-CL3	IAR	IAR Embedded Workbench for Arm V8.42.1 ~ Latest Version (Please use the latest version) (IAR Systems)			I-Jet I-jet Trace for Arm Cortex-M (IAR Systems)

Table 1.2 CMSIS version

	R-IN32M4-CL3
Version	V4.5.0

1.3.2 Evaluation board

Operation of the TCP/IP stack sample application can be confirmed on the "R-IN32M4-CL3 evaluation board" manufactured by Shimafuji Electric Incorporated and the "TS-R-IN32M4-CL3 evaluation board" manufactured by TESSERA TECHNOLOGY INC.

Please get more detail from Renesas or Shimafuji Electric Incorporated WEB site.

1.3.3 Development procedure

Standard development procedure is as follows:

1. Merge TCP/IP stack program and samplesoft which has driver/middleware.
2. Modify net_cfg.c to configure network parameter (IP address, socket definition) and how to call initialize routine.
3. Make the execution file by build (compile & link), after application program is created.

The relationship among files is showed at Fig. 1.1.

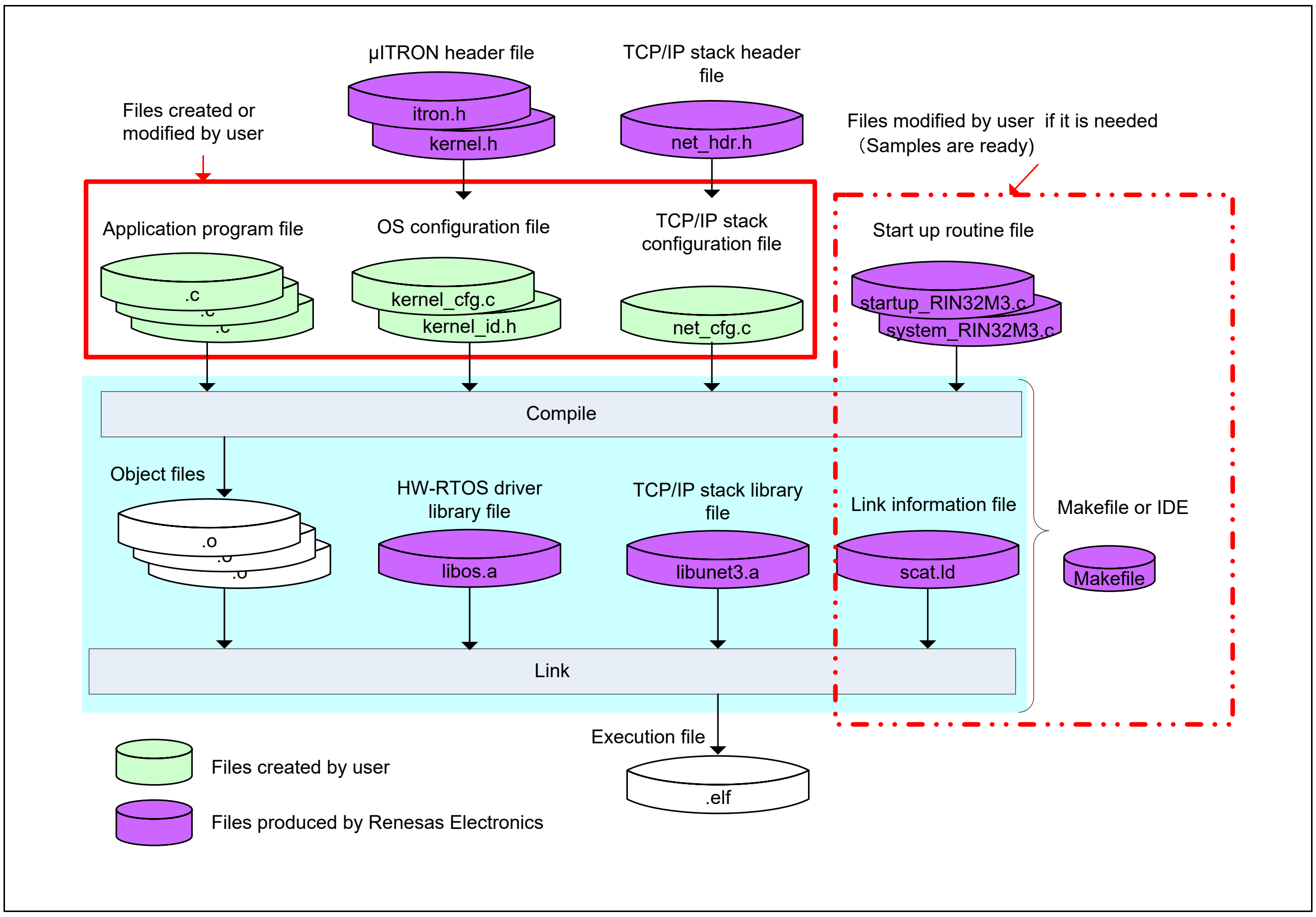


Fig. 1.1 The figure of relationship among files

2. Basic concepts of TCP/IP stack

2.1 Glossary

2.1.1 Protocol

Protocol is a set of rules that determines the method and the procedure of transmitting data between networks. TCP/IP stack adopts this protocol (=communication rule). These rules are called Request For Comments (abbreviation: RFC)", its specification is published.

2.1.2 Protocol stack

Choose a necessary protocol in order to implement functions on network, and a protocol stack is a prescribed hierarchy of software layers. The following figure shows the hierarchy in TCP/IP stack.

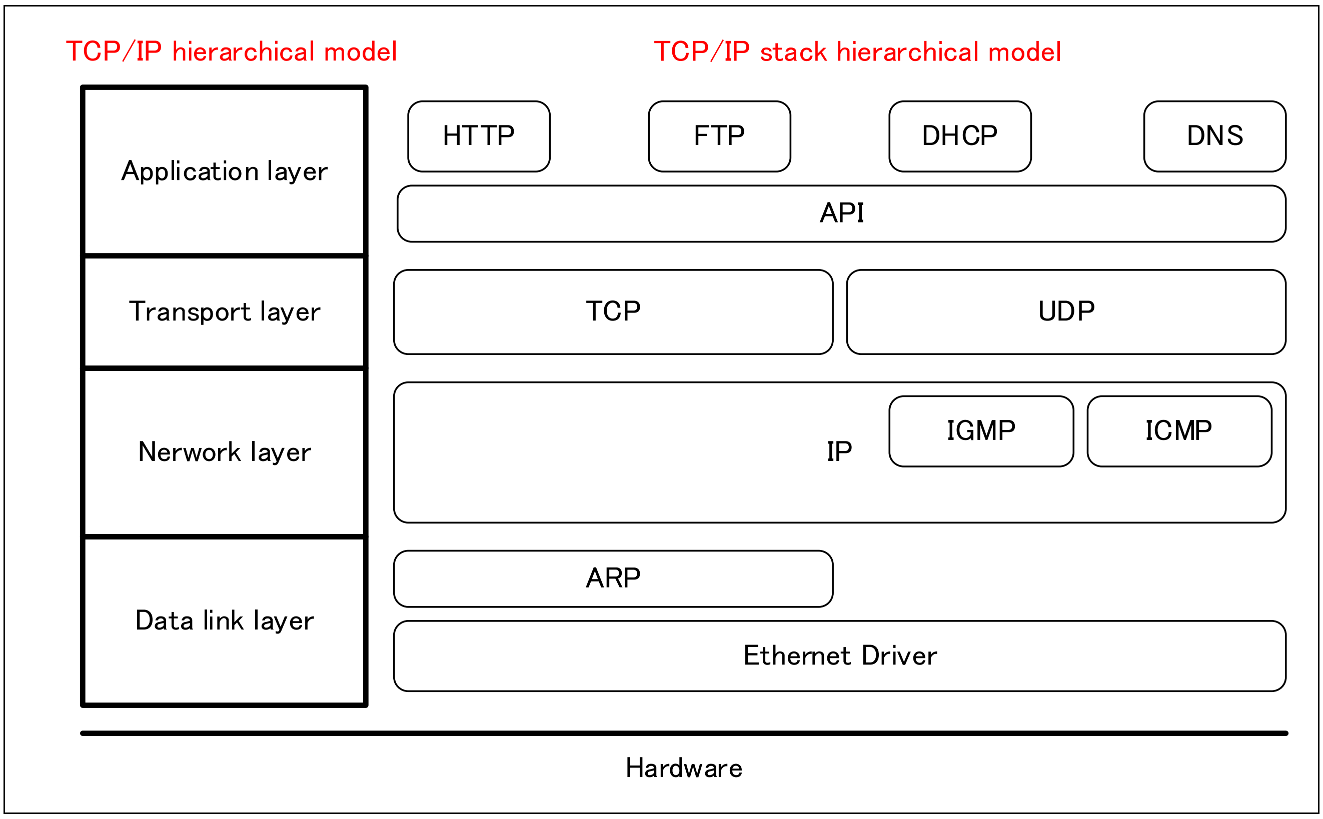


Fig. 2.1 Figure of TCP/IP hierarchical model and TCP/IP stack hierarchical model

2.1.3 IP address

Each node on the network has a specific logical number, it is called “IP address”. IP address has 32 bit address space, be represent as 192.168.1.32.

(1) Broadcast address

Broadcast means that the same data is simultaneously sent (broadcast communication) to all of the nodes in one network. The address is allocated particularly to broadcast called “Broadcast address”. Ordinarily in “Broadcast address”, all bits use 1 IP address “255.255.255.255”.

(2) Multicast address

Contrary to the broadcast that send data to all nodes, a special address is used to send data to a specific group only, is called “Multicast address”.

2.1.4 MAC (Media Access Control) address

Contrary to a logical address “IP address”, a physical address specify to an installed hardware in order to identify network devices such as LAN card is called “MAC address”. “MAC address” has 48-bit address space and to be notated 12-34-56-78-9A-BC or 12:34:56:78:9A:BC.

2.1.5 Port number

In network communication, a number identifies a program of communication partners is called “Port number”. The node that communicate through TCP/IP has IP address that corresponds to the address inside the network, but in order to communicate with more than one node at the same time, we use port number in the range from 0 to 65535 as auxiliary address.

2.1.6 Big endian and little endian

The way multibyte numerical data is stored in memory is called “Endian”. “Big endian” refers to the way that store the most significant byte in the sequence. “Little endian” refers to the way that store the least significant byte in the sequence

It is determined that the header information is transmitted by “big endian” through TCP/IP.

2.1.7 Packet

The Unit of data transceiver is called “packet”. The packet includes 2 kinds of information. One contains actual stored data (data area) and the other contains the information used to manage as the information of source or destination of that data, error checking information (header area).

2.1.8 Host and node

Host refers to the computer that communicates on the network. And the connection points in a network such as server, client, hub, router, access point etc. are called “node”.

2.1.9 Address Resolution Protocol (ARP)

A protocol used to translate the physical address (MAC address) from logical address (In case of TCP/IP, that is IP address) is called “ARP”.

2.1.10 Internet Protocol (IP)

The protocol which executes the communication between nodes or node and gateway is called “IP (IP protocol)”. “IP (IP protocol)” is an very important protocol of the upper layer. The role of “IP” is to transfer data to the destination through the router based on the IP address without however ensuring their delivery, thus, ensuring the reliability of data is upper layer’s responsibility

“IP address” mentioned above is placed in the header of this “IP protocol”.

2.1.11 Internet Control Message Protocol (ICMP)

A protocol provides the function that is to notify errors occurred in IP network communication and verify the state of network status is called “ICMP”. There are echo request and echo reply messages are called Ping which most well-known.

2.1.12 Internet Group Management Protocol (IGMP)

The protocol executes IP Multicast is called “IGMP”. We can usually send the same data to many different hosts efficiently.

2.1.13 User Datagram Protocol (UDP)

A protocol provides the connectionless mode datagram communication service is called “UDP”. IP does not have interface with application. “UDP” is the protocol which helps to use that function from application. As a result, there is no way to notify that packets have arrived to the partner and the order of arrived packets may be changed so UDP does not sure the reliability of data.

2.1.14 Transmission Control Protocol (TCP)

A protocol which provides connection mode stream communication service is called “TCP”. “TCP” is known as upper layer of IP protocol, which provides a reliable communication as flow control, retransmission, error correction and sequence control.

2.1.15 Dynamic Host Configuration Protocol (DHCP)

When connecting to a network, a protocol which assigns automatically the necessary information such as IP address is called “DHCP”. To use “DHCP”, we have to prepare DHCP server and on server side, it’s necessary to prepare some IP addresses for DHCP client in advance (Address pool).

2.1.16 Hyper Text Transfer Protocol (HTTP)

A protocol used to transfer the contents such as HTML file of homepage or website is called “HTTP”. “HTTP” not only can transfer HTML file but also can send binary data which are displayed on WEB browser such as JPEG, GIF, PNG, ZIP file.

2.1.17 File Transfer Protocol (FTP)

We call the protocol which transfer files between hosts is “FTP”.

2.1.18 Domain Name System (DNS)

A name resolution mechanism which can exchange host name into IP address or IP address into host name (domain) is called DNS. In case of using “DNS”, it is possible to look up the host name based on IP address or look up IP address from the host name.

2.1.19 Socket

An endpoint for communication which applications use for communicating TCP/IP is called “socket”. “Socket” is constructed by IP address and port number. The applications, through specifying the socket to establish a connection, can transceive data without caring about any details of communication procedure. There are varieties of sockets depending on the protocol used in communication side. TCP socket uses TCP protocol to communicate data and UDP socket uses UDP protocol to communicate data. In TCP/IP stack, we use ID number to identify the socket which becomes an operational objective. The application utilizes ID number to invoke socket API.

2.1.20 Blocking and non-blocking

When calling some function, if it does not return until the action has completed, that is called “Blocking mode” and if it returns immediately without waiting for its completion, that is called “Non-blocking mode”.

For instance, in the socket API ofμNe3, the task calling the rcv_soc function in “Blocking mode” is placed in the waiting status until that action completes (until data can be received). Calling the rcv_soc function in “Non-blocking mode” will return immediately with an E_WBLK error code and the completion of that action (EV_RCV_SOC) is notified to callback function.

By default, the TCP/IP stack sockets are in “Blocking mode”, and in order to switch to “Non-blocking mode”, we have to use cfg_soc function and set up registration of callback function and callback event flag.

2.1.21 Callback function

The function used for notifying the status of protocol stack to the applications asynchronously is called “Callback function”.

2.1.22 Task context

All API functions of TCP/IP stack must be called from Task context.

Do not call the system call which is in status of waiting for tasks such as slp_tsk from network callback function. Besides, do not call all API functions of TCP/IP stack from network callback function.

2.1.23 Resource

The resource used in a program is called “Resource”. There include tasks, semaphores are called “Kernel objects” and memory.

※Please make reference to OS User Guide on details of “Kernel object” such as tasks, semaphores.

2.1.24 MTU

In communication network, MTU (Maximum Transfer Unit) is a value indicating the maximum amount of data that can be transferred by one-time transfer. Moreover, MTU also shows the maximum data size of the frame in data link layer. In addition, the minimum value indicated by MTU is 68 bytes.

Specifying the maximum data size depends on the protocol used in data link layer and in Ethernet interface generally uses 1500 bytes.

2.1.25 MSS

MSS (Maximum Segment Size) indicates the maximum data size of TCP packet. Therefore, MSS value can be calculated by the following formula.

MSS = MTU– (IP Header size+ TCP Header size (normally 40 bytes))

In case of Ethernet interface, the value of MSS is generally 1460 bytes.

2.1.26 IP reassembly - fragment

The maximum size of an IP packet is 64K bytes. However, in order that MTU of communication interface becomes a smaller value than the original, it’s necessary that IP module must divide IP packet into smaller pieces to send. This processing is called “IP fragmentation” and divided IP packet is called “IP fragment”.

Moreover, IP module of receiver side needs to combine the divided “IP fragment” and we call this process is “IP reassembly”.

2.2 Architecture of Network system

2.2.1 Block diagram of network system

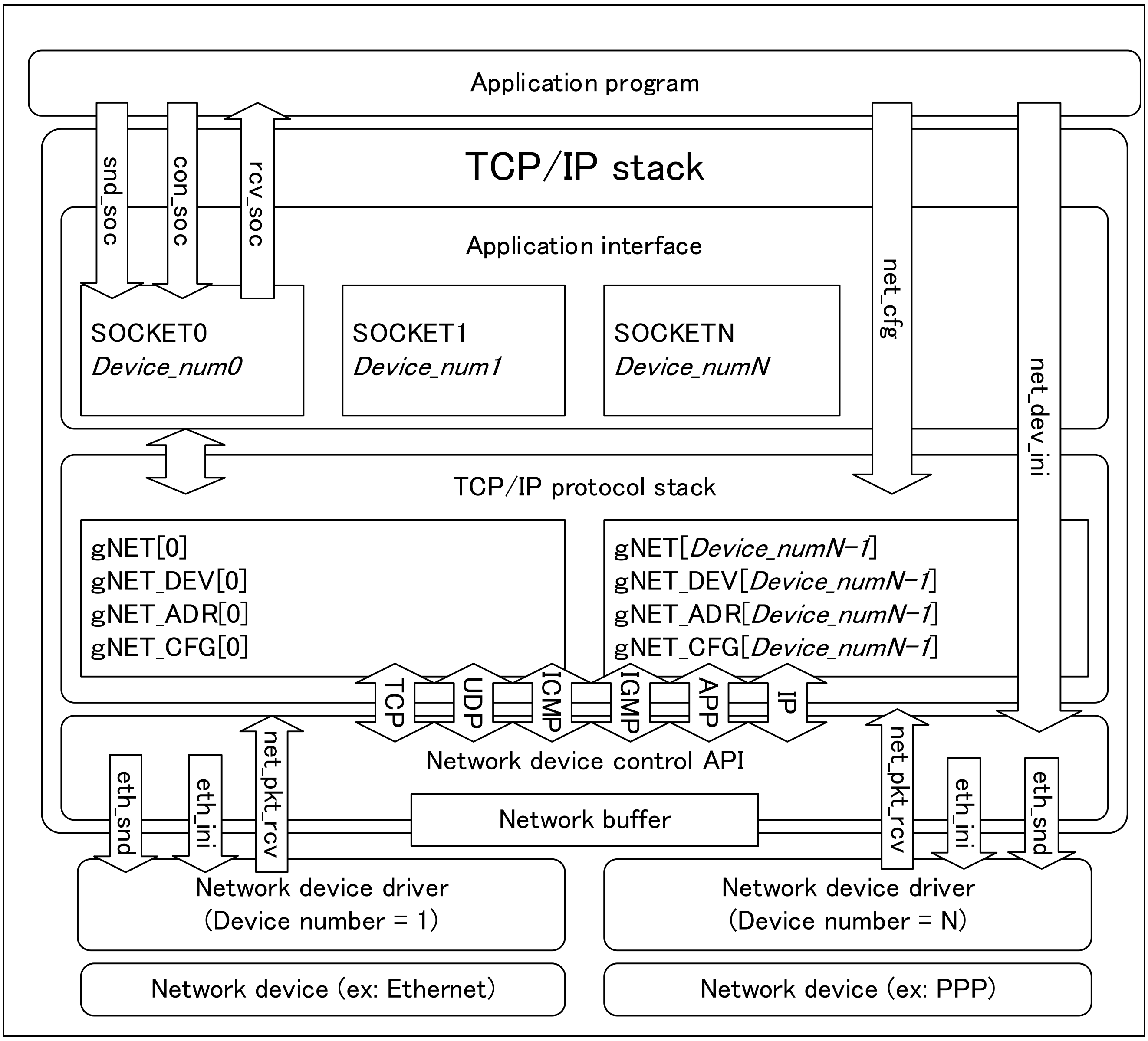


Fig. 2.2 Block diagram of network system

- Application program

The application program is used for network communication. It includes application protocols such as DHCP, FTP, Telnet, HTTP etc.

- Application Interface

Providing the interface (API) to utilize various network services such as transmission / reception of data or establishing connection to remote host.

In case of normal application, we have to specify socket ID and device number before using Application Interface.

- TCP/IP protocol stack

This program handles the network protocols such as TCP, UDP, ICMP, IGMP, IP and ARP.

- Network device control API

In network system, maybe there exist various network devices. Every device needs a device driver. And the network device control API absorbs the difference between these devices, provides interface in order to access unifiedly. Using device number from application program before accessing to the devices.

- Network device driver

A program that control network device. The content integration is different depending on the device.

In TCP/IP stack, it is provided standard Ethernet driver device.

- Network device

The hardware that execute the transmission and reception of actual network data. This refers to Ethernet, PPP (RS-232), and WLAN etc.

2.3 Directory and file organization

These files as below are included in TCP/IP stack.

(1) Header File

/Source/Middleware/uNet3/Inc/

net_sup.h	Default configuration macro for TCP/IP protocol stack
net_def.h	TCP/IP protocol stack definition (for internal control)
net_sts.h	Definition of network information management (for internal control)
net_sts_id.h	Definition of network information management ID
net_hdr.h	Definition of the necessary information to use TCP/IP protocol stack Please include this header file in source file of applications.

(2) Library files

This folder stores the library that has already built the TCP/IP protocol stack in various processor mode, and the project files are used to build.

/Library/IAR/

libunet3.a	TCP/IP protocol stack library
libunet3bsd.a	TCP/IP protocol stack library (BSD interface)
libunet3snmp.a	TCP/IP protocol stack library (SNMP interface)

(3) Ethernet device driver file

/Source/Driver/ether_uNet3/

DDR_ETH.c	Ethernet driver
DDR_PHY0.c	PHY driver for LAN1
DDR_PHY1.c	PHY driver for LAN2

/Source/Driver/ethsw/

ethsw.c	Ethernet Switch driver

/Include/ ether_uNet3/

DDR_ETH.h	Ethernet driver header
DDR_PHY.h	PHY driver common header
COMMONDEF.h	Ethernet driver common definition header

/Include/ ethsw/

ethsw.h	Ethernet Switch driver header

(4) Application protocol source file

/Source/Middleware/uNet3/NetApp/

dhcp_client.h	DHCP client macro, prototype, definition etc.
dhcp_client.c	DHCP client source code
ftp_server.h	FTP server macro, prototype, definition etc.
ftp_server.c	FTP server source code
http_server.h	HTTP server macro, prototype, definition etc.
http_server.c	HTTP server source code
dns_client.h	DNS client macro, prototype, definition etc.
dns_client.c	DNS client source code
ping_client.h	ICMP echo request macro, prototype, definition etc.
ping_client.c	ICMP echo request (ping) source code.
sntp_client.h	SNTP client macro, prototype, definition etc.
sntp_client.c	SNTP client macro source code.
net_strlib.h	String library function definition.
net_strlib.c	String library function source code.

/Source/Middleware/uNet3/NetApp/ext/

dhcp_client.h	DHCP client extended version macro, prototype, definition etc.
dhcp_client.c	DHCP extended version client source code

/Source/Middleware/uNet3/NetApp/cfg/

ftp_server_cfg.c	FTP server configuration
ftp_server_cfg.h	FTP server configuration header
http_server_cfg.h	FTP server configuration header

/Source/Middleware/uNet3/snmp/

inc/snmp.h	SNMP Headers
inc/snmp_ber.h	SNMP BER
inc/snmp_def.h	SNMP constants, prototypes, definitions, etc.
inc/snmp_lib.h	SNMP Library Configurations
inc/snmp_mac.h	SNMP Macro Definition
inc/snmp_mib.h	SNMP MIB definition
inc/snmp_net.h	Definition for SNMP TCP/IP
src/snmp_mib_dat.c	SNMP MIB definition

(5) Sample source file

/Source/Project/uNet3_sample/

cgi_sample.c	CGI application sample
DDR_ETH_CFG.h	Ethernet driver configuration header
html.h	HTML data
kernel_cfg.c	OS resources configuration file
kernel_id.h	OS resources ID definition file
main.c	Main function
net_cfg.c	TCP/IP stack configuration file
net_sample.c	Network application sample

/Source/Project/uNet3_nonblock/

DDR_ETH_CFG.h	Ethernet driver configuration header
nonblock_sample.c	Echo server sample that works non-blocking
kernel_cfg.c	OS resources configuration file
kernel_id.h	OS resources ID definition file
main.c	Main function
net_cfg.c	TCP/IP stack configuration file
net_sample.c	Network application sample

/Source/Project/uNet3_bsd/

DDR_ETH_CFG.h	Ethernet driver configuration header
socket_command.c	Operation check program using a serial console
kernel_cfg.c	OS resources configuration file
kernel_id.h	OS resources ID definition file
main.c	Main function
net_cfg.c	TCP/IP stack configuration file
net_sample.c	Network application sample

/Source/Project/uNet3_mac/

console.c	Test program using serial console
DDR_ETH_CFG.h	Ethernet driver configuration header
kernel_cfg.c	OS resources configuration file
kernel_id.h	OS resources ID definition file
main.c	Main function
net_cfg.c	TCP/IP stack configuration file
net_sample.c	Network application sample

/Source/Project/uNet3_snmp

cgi_sample.c	CGI application sample
DDR_ETH_CFG.h	Ethernet driver configuration header
html.h	HTML data
kernel_cfg.c	OS resources configuration file
kernel_id.h	OS resources ID definition file
main.c	Main function
net_cfg.c	TCP/IP stack configuration file
net_sample.c	Network application sample
snmp_cfg.c	SNMP configuration file
snmp_mib_cfg.c	MIB definition file

3. Overview functions of TCP/IP stack

3.1 Protocol stack

3.1.1 IP module

The IP module only receives and handles the arrived packets which has destination IP address matches with the IP address of local host. Other packets are not handled.

(1) IP Option

TCP/IP stack supports router warning option of internal IGMP in IP option only. IP options which do not support will be ignored. .

(2) TTL (Time to Live)

Default value of TTL in TCP/IP stack is set CFG_IP4_TTL(64). This value may be changed by using net_cfg(). In case of using net_cfg() to change the value of TTL, TTL value of all sockets are changed. In case that we want to change TTL value of each socket, please use cfg_soc().

(3) TOS (Type Of Service)

In TCP/IP stack, TOS is set CFG_IP4_TOS (0).

(4) Broadcast

Maybe receive broadcast or not depending on using net_cfg(). The initial value is set that ready to not receive. Always can transmit broadcast. The broadcast setting is effective for all sockets but we cannot set up whether receive broadcast by socket unit.

Regarding to transceive broadcast, please use UDP socket.

(5) Multicast

In order to allow multicast reception, we use net_cfg() and register at the address of the multicast group which join to. Multicast group address may be registered by CFG_NET_MGR_MAX (8).

Always can send multicast. The multicast setting is effective for all sockets but we cannot set up whether receive multicast by socket unit.

TL used for transferring multicast is set CFG_IP4_MCAST_TTL (1). This value can also be changed by using net_cfg().

Do not support multicast loopback.

Regarding to transceive of multicast, please use UDP socket.

(6) MTU

In TCP/IP stack, CFG_PATH_MTU (1500 byte) is set as default value of MTU. This value can be configured by the configurator.

(7) IP reassembly / fragment

In TCP/IP stack, maximum size of IP packet is 1500 byte as default (This value is related to the value of network buffer). In order to increase the size of IP packet to maximum, we need to enlarge the network buffer. For example, in case that transceive 2048byte of UDP data, we need to increase the value of network buffer larger than the value is calculated from this formula (control header size (100 bytes) + IP header size (20bytes) + UDP header size (8bytes) + 2048).

The default value of IP reassembly process timeout is CFG_IP4_IPR_TMO(10 seconds). If the reassembly process cannot complete within this timeout, the reassembly process is cancelled, the ICMP error message (type 11: packet discarded by time excess) is sent to remote host.

The default number of times of the IP reassembly process is set CFG_NET_IPR_MAX(2). CFG_NET_IPR_MAX value expresses a value which host can execute IP reassembly process at the same time.

(8) IGMP

In TCP/IP stack, the timeout until the “report (reply)” message is sent to “query (group inquiry)” (from router) is set by CFG_IGMP_REP_TMO (10 seconds)

TCP/IP stack supports IGMPv2 and also supports IGMPv1 compatible function.

In case of getting query of IGMPv1, it will be changed into IGMPv1 mode and then processed. After that, within a certain time period, if there is no IGMPv1 message, it will be back to IGMPv2 mode. Timeout for returning from IGMPv 1 to IGMPv 2 is set by CFG_IGMP_V1_TMO (400 seconds).

(9) ICMP

TCP/IP stack supports messages of “echo response”, “echo request”, “time excess”.

3.1.2 ARP module

(1) Resolve ip address

TCP/IP stack will manage the mapping of IP address of host and physical address (MAC address). The administration table (conversion table) of this mapping is called ARP cache. ARP cache size is set by CFG_NET_ARP_MAX (8).

When sending IP packet to network, in case that there exists a compatible IP address which refers to ARP cache, it will send a packet to the destination that is the physical address has been recorded there. In case that there is no existing IP address, IP packet will be stored temporarily in queue, then, send broadcast ARP request packets. After receiving ARP response packets from remote host, record a received physical address in ARP cache newly. Then, remove IP packet from queue, send the packet to the newly acquired physical address.

Besides, ARP entry information is held in the cache table for a maximum of ARP_CLR_TMO (20 minutes).

(2) Address conflict detection

According RFC5227, TCP/IP stack will check whether Ipv4 address is non-duplicative in the same link. This feature is performed by API is called from application, when LAN interface boot up or link status changes.

After setting the IP address of the interface, the other host had set the same IP address, then the detected the conflict, TCP/IP stack will notify the application.

"ARP Probe" can detect whether the IP address that you will use is not already in use.

Other host did not respond to **"ARP Probe"**(IP address conflict is not), TCP/IP stack notify the other hosts that to use this IP address from now by sending the **"ARP Announce"**.

3.1.3 UDP module

UDP executes the transceiver of data without connecting to remote host.

(1) Sending data

Before sending data, we should use con_soc and associate a socket with a source address (IP address, port number). After that, we use snd_soc() to send data. The flow snd_soc() processing is described in the diagram as below.

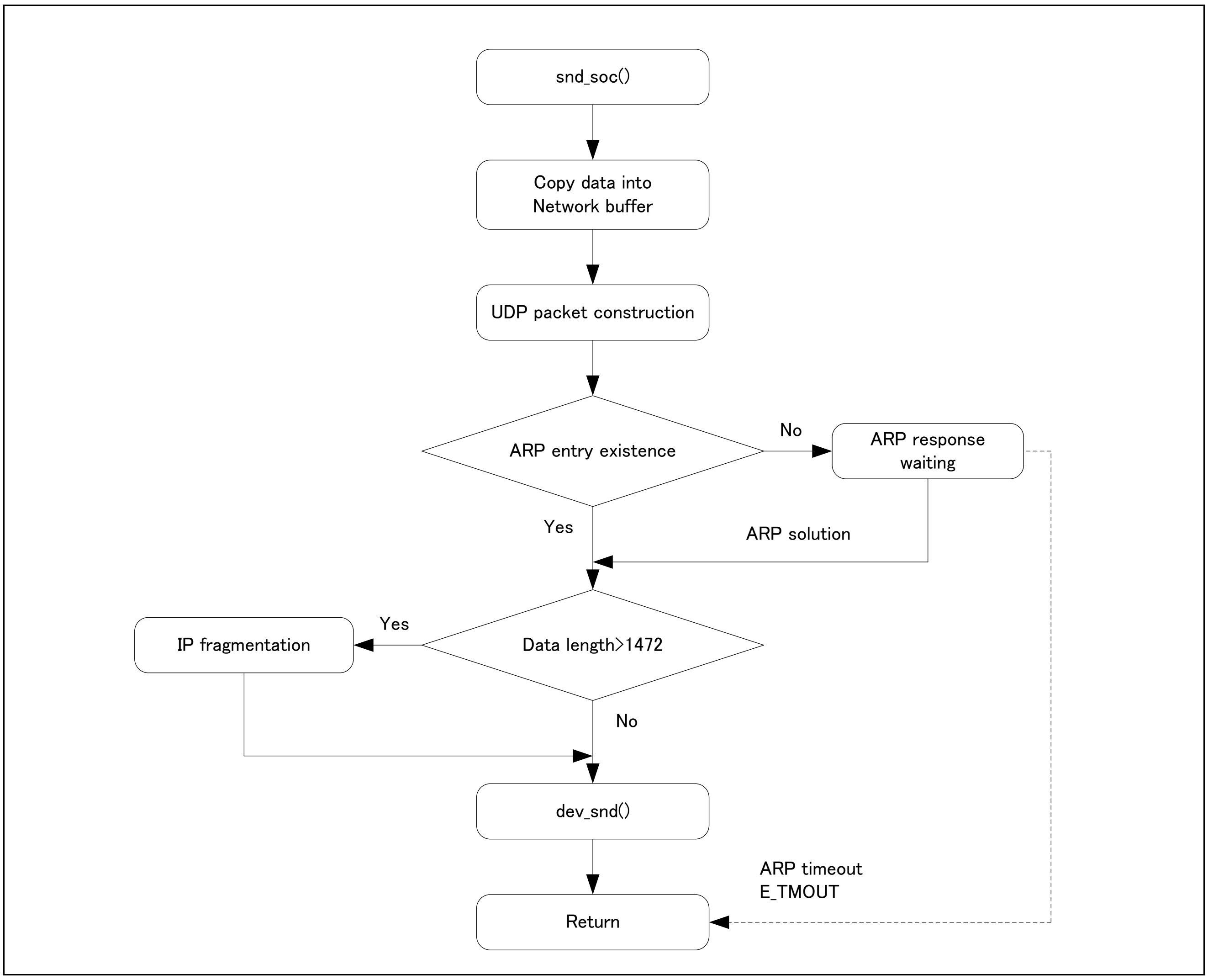


Fig. 3.1 The flow of snd_soc processing of UDP socket

4. The application data is copied into network buffer, adding UDP header such as port number,IP address of remote host,then construct UDP packet.
5. In case that cannot resolve MAC address of remote host by ARP protocol, it will return E_TMOUT error.
6. In default, the maximum size of transmission data is set 1472 bytes (CFG_PATH_MTU (1500 bytes) – IP header size– UDP header size). In case of sending data with larger size than this, we need to set network buffer size. Regarding the details, please refer to the item of IP reassembly/fragment

(2) Data reception

Data reception is executed by using rcv_soc(). The flow of rcv_soc() processing is described in the diagram as below.

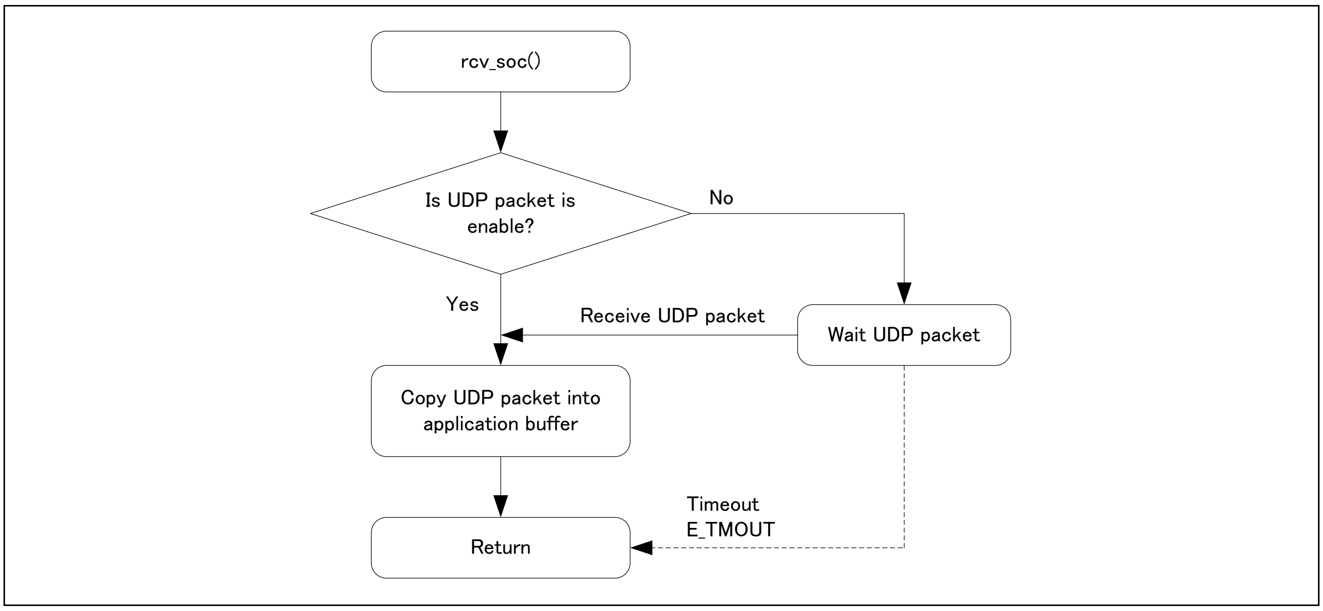


Fig. 3.2 The flow of rcv_soc processing of UDP socket

7. If UDP packet has not been received yet, enter a state of waiting for UPD packet reception. At that time, if it exceeds timeout of receiving socket, it will return E_TMOUT.
8. If received packet size is smaller than requested data size, copy into application buffer. In case that received packet with bigger size, just copy the request size into application buffer. Remaining part will be ignored.
9. In default, maximum size of reception data is set 1472 bytes (CFG_PATH_MTU (1500 bytes) – IP header size- UDP header size). In case of receiving data with larger size than this, we need to set network buffer size. Regarding the details, please refer to the item of IP reassembly/fragment.

3.1.4 TCP module

TCP is different from UDP. TCP is connection mode, so it can allocate sending party and channels before transceiving data. TCP sequence is described in the diagram as below.

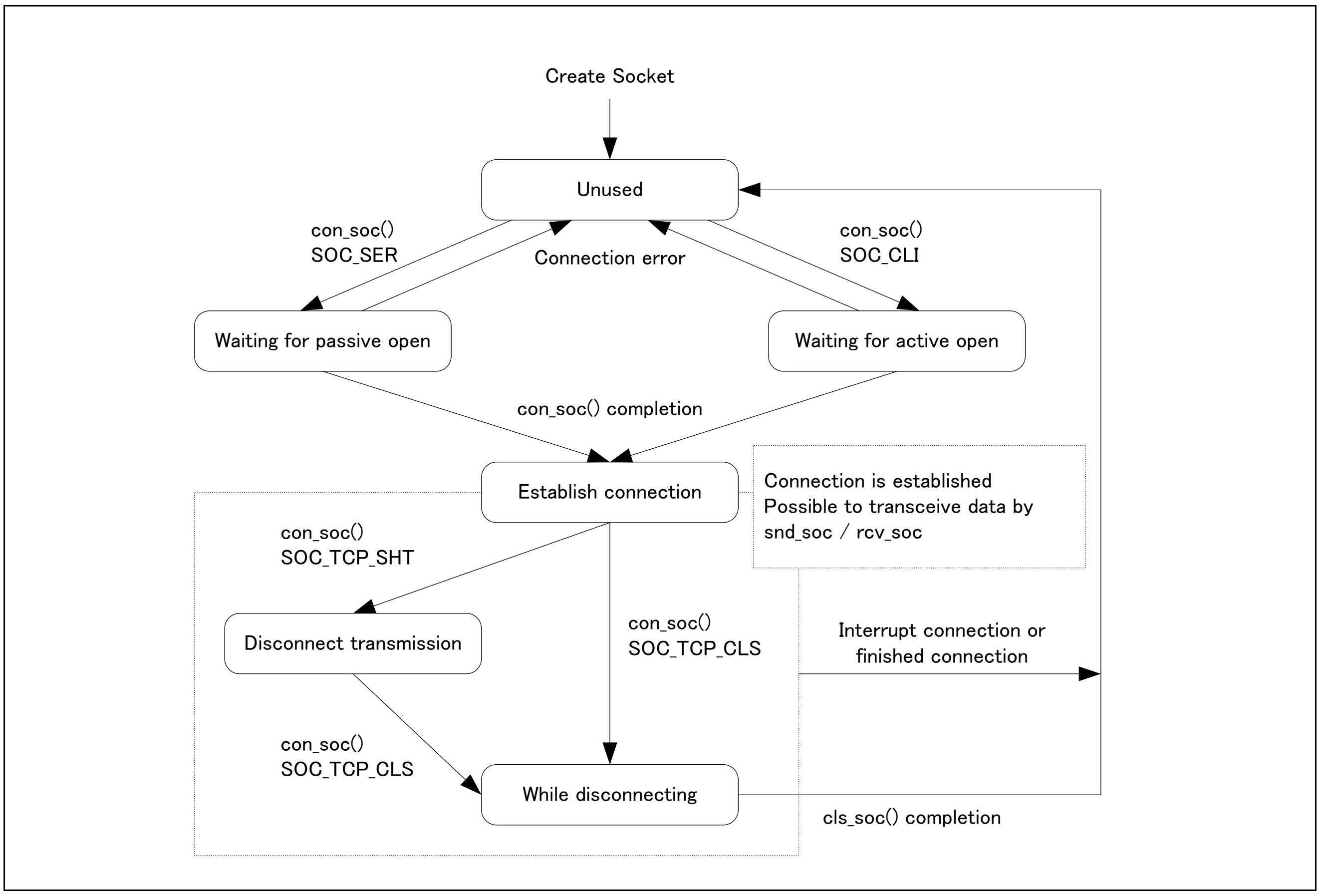


Fig. 3.3 TCP sequence

(1) Establishing connection

There are two modes of TCP connection, active and passive connection. Active connection that requires connect to remote host by itself. On the contrary, passive connection that wait for the connection from remote host.

Use con_soc() to connect, and need to specify active connection by SOC_CLI and passive connection by SOC_SER.

(2) Connection completion

In order to disconnect the connection, we use cls_soc(). Specify SOC_TCP_CLS in order to disconnect all the connection completely, and SOC_TCP_SHT to disconnect the transmit direction only.

(3) Data transmission

Use snd_soc() to send data. The flow of snd_soc() processing is described as below.

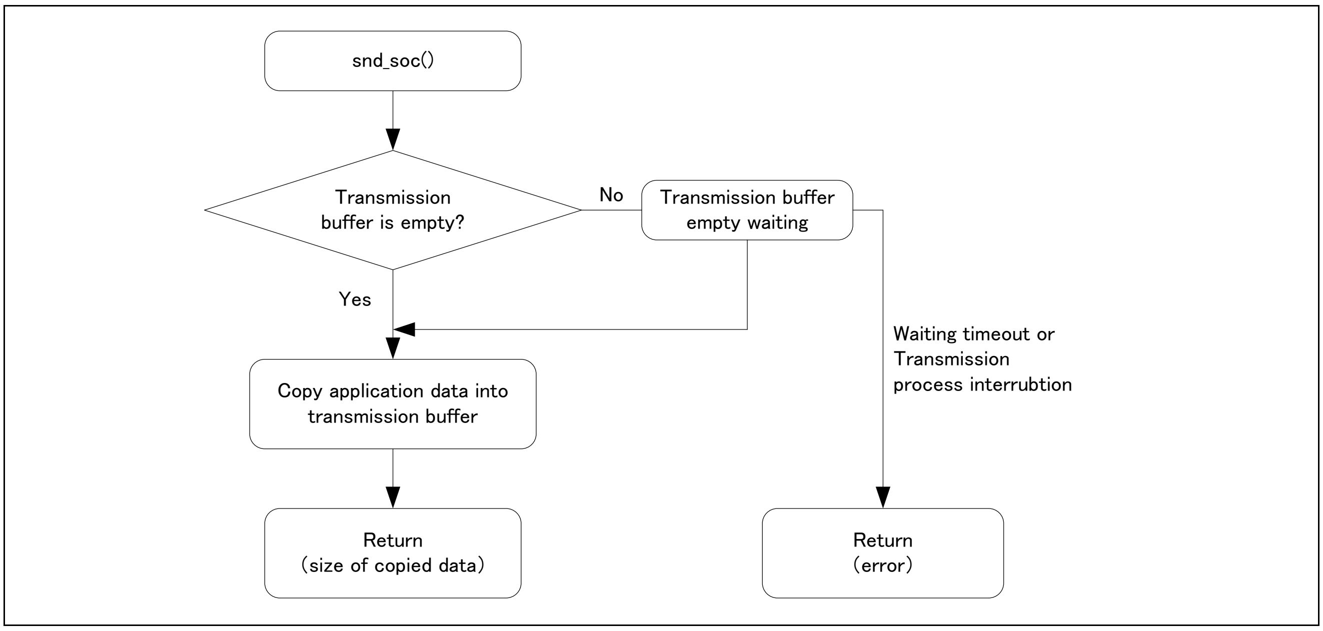

snd_soc()
Transmission buffer is empty?
No
Transmission buffer empty waiting
Yes
Waiting timeout or Transmission process interrubtion
Copy application data into transmission buffer
Return (size of copied data)
Return (error)

Fig. 3.4 TCP socket–Flow of snd_soc processing

10. Copy data of application into TCP transmission buffer. If copy is successful, TCP protocol will send data. If remote host received data, all data in TCP transmission buffer will be clear.

- TCP transmission buffer

It is necessary to specify transmission buffer size when create TCP socket. Buffer size ranges from 4 bytes to 32 kilobytes and is specified as a power of 2 (1024, 2048, 4096, and so on).

(4) Data reception

Use rcv_soc() to send data. Received TCP packet firstly will be registered at TCP reception buffer. When rcv_soc() is called, it will be copied from TCP reception buffer into application buffer.

- TCP reception buffer (Window buffer)

It is necessary to specify reception buffer size when create TCP socket. Buffer size ranges from 4 bytes to 32 kilobytes and is specified as a power of 2 (1024, 2048, 4096, and so on).

(5) Retransmission timeout

Timer sequence of resending is described in the diagram as below.

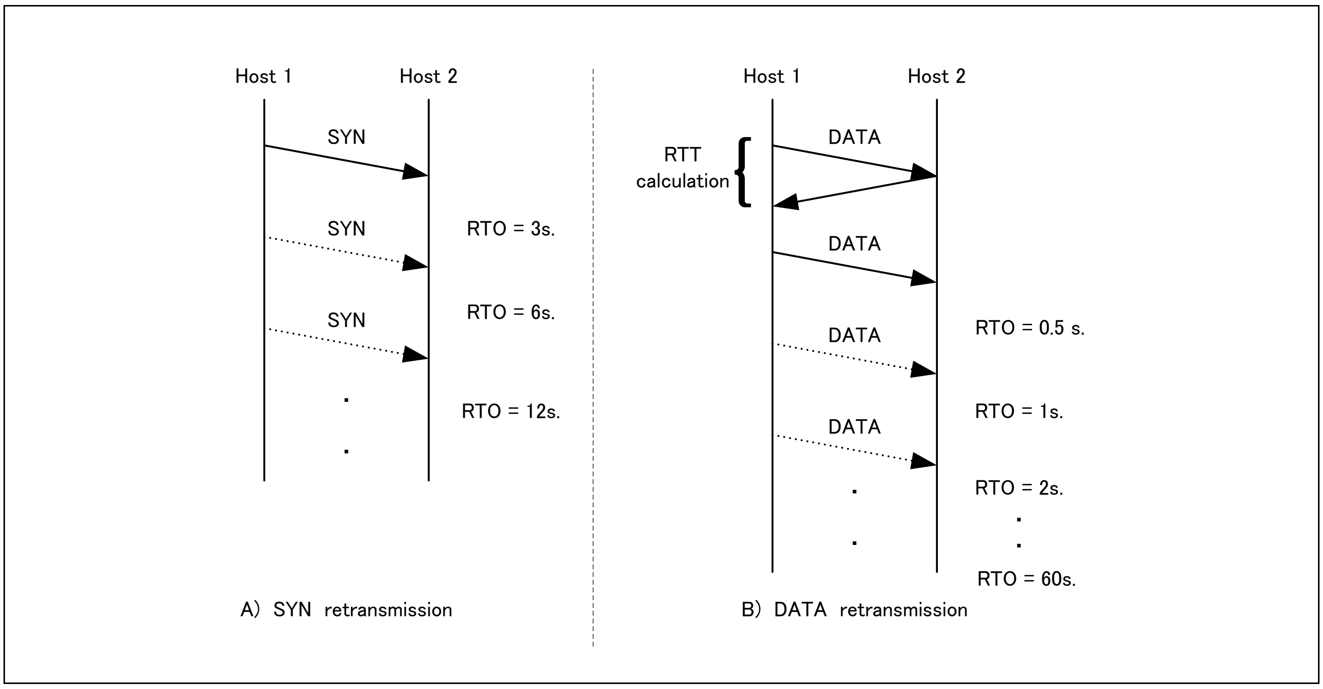


Fig. 3.5 An example of retransmission timer

In TCP, if there are not response of ACK packet within a certain time for any reason, segment without response will be sent again. The waiting time until retransmission action is executed is called "RTO" (Retransmission Time Out). Initial value of RTO is called "RTT" (Round Trip Time), is "4 times＋α" of "Time that packet makes round trip to the other). RTO value is increased twice every time resending action is done.

When retransmit SYN like the above A diagram, it uses CFG_TCP_RTO_INI (3 seconds) due to RTT value is not set. In the above B diagram of data retransmission, it calculates RTT value based on the previous successful transmission, that's 500 milliseconds.

RTO scope is set from CFG_TCP_RTO_MIN (500 ms) to CFG_TCP_RTO_MAX (60 s).

(6) Connection timeout

Connection timer sequence is described in the below diagram.

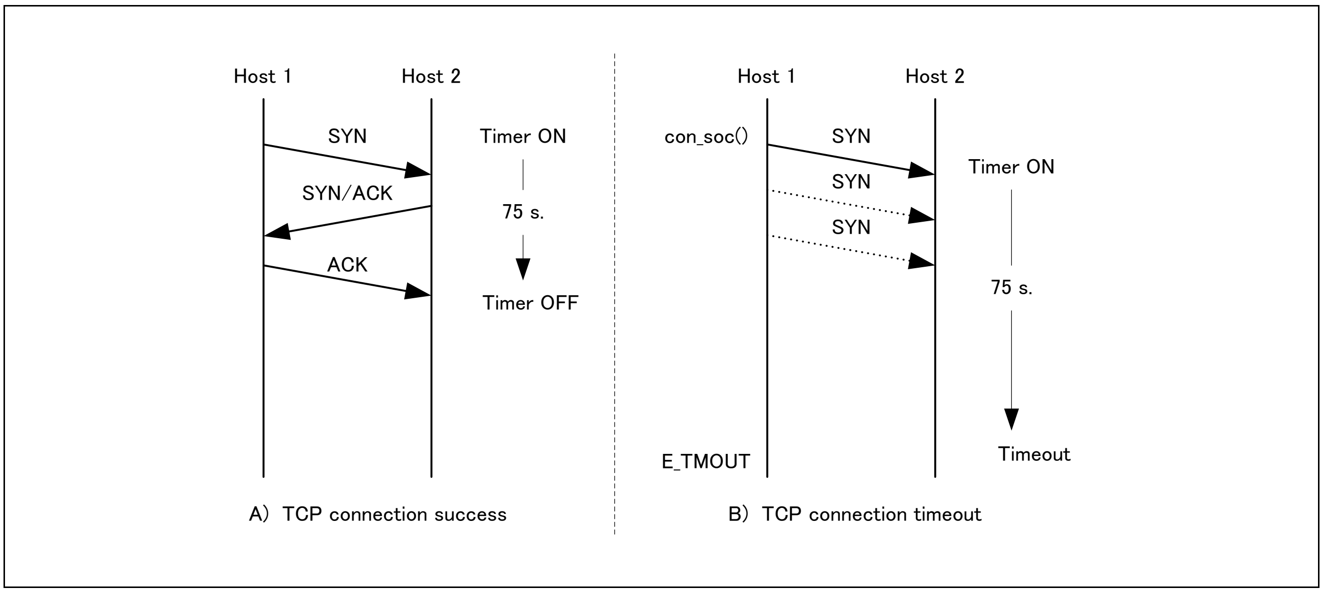


Fig. 3.6 An example of connection timeout

When call con_soc(), if this timer completes from starting up to three-way handshake timed out, it will return E_OK (A). If finish timeout, it will return E_TMOUT (B).

Timeout value of connection process (3-way handshake) is set CFG_TCP_CON_TMO (75 seconds).

※When create TCP socket, it can specify blocking timeout used in connection. If this value runs out of time, connection process will be interrupted immediately and con_soc() will return E_TMOUT.

(7) Transmission timeout

Transmission timeout is set CFG_TCP_SND_TMO (64 seconds). While communicating data, if there is no response from the partner even though passes CFG_TCP_SND_TMO, the connection will be disconnected.

(8) Disconnection timeout

Timeout of disconnection process is set CFG_TCP_CLS_TMO (64 seconds). If cls_soc() does not complete at CFG_TCP_CLS_TMO, connection will be forcibly disconnected and cls_soc() will return E_TMOUT.

※When create TCP socket, it can specify blocking timeout used in connection. If this value runs out of time, connection process will be interrupted immediately and cls_soc() will return E_TMOUT.

(9) Delay ACK timeout

Delay ACK timeout is set CFG_TCP_ACK_TMO (200 milliseconds).

(10) TCP congestion control

TCP/IP stack supports fast retransmit and fast recovery. Number of duplicate ACK is set CFG_TCP_DUP_CNT (4).

(11) Maximum Segment Size (MSS)

MSS is set CFG_TCP_MSS (1460 bytes).

(12) Keep Alive

TCP/IP stack supports TCP Keep Ailve.

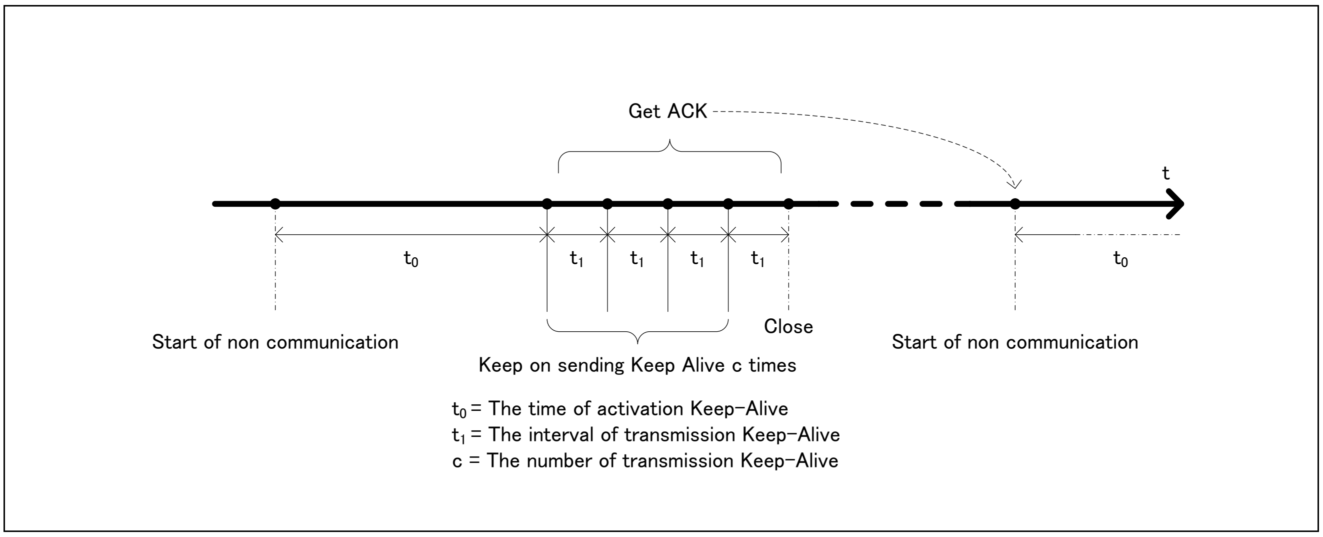

Get ACK
t
t_0
t_1
t_1
t_1
t_1
t_0
Start of non communication
Close
Start of non communication
Keep on sending Keep Alive c times
t_0 = The time of activation Keep−Alive
t_1 = The interval of transmission Keep−Alive
c = The number of transmission Keep−Alive

Fig. 3.7 Operation TCP Keep Alive

If Keep Alive feature is enabled ($c > 0$), after t_0 seconds in non-communication state, start the transmission of Keep Alive packet to the destination host.

Until it is transmitted c times, or get ACK from destination, TCP/IP stack will continue to send the Keep Alive packets at intervals of t_1 seconds.

If no response is obtained in c times Keep Alive packet, then TCP/IP stack close TCP connection.

It does not disconnect TCP connection automatically if Keep Alive feature is disable ($c = 0$)

3.2 Network device driver

Because protocol stack will access to device driver through **T_NET_DEV structure**, so the information as device name, device number, the functions of device driver must be registered in **T_NET_DEV structure** in advance. Protocol stack specify and access to the device which is registered in **T_NET_DEV** by device number.

3.2.1 Device structure

```
typedef struct t_net_dev {
        UB          name[8];        /* Device name*/
        UH          num;            /* Device number */
        UH          type;           /* Device type */
        UH          sts;            /*Reserve*/
        UH          flg;            /*Reserve */
        FP          ini;            /*Pointer to dev_ini function*/
        FP          cls;            /* Pointer to dev_cls function*/
        FP          ctl;            /* Pointer to dev_ctl function*/
        FP          ref;            /* Pointer to dev_ref function*/
        FP          out;            /* Pointer to dev_snd function*/
        FP          cbk;            /* Pointer to dev_cbk function*/
        UW          *tag;           /* Reserve*/
        union       cfg;            /* MAC address */
        UH          hhdrsz;         /* Device header size */
        UH          hhdrofs;        /* Position writing network buffer */
        VP          opt;            /* Driver extension area */
} T_NET_DEV;
```

(1) Device number

Set unique number to specify device. Protocol stack will use this number to access to the device. Device number should be numbered consecutively from 1.

(2) Device name

Set the name in order to specify the device. The length of device name should be under 8 bytes long.
For example: eth0, eth1 etc.

(3) Device type

Set type of network device. There are some as below.

Device type	Meaning
NET_DEV_TYPE_ETH	Ethernet device
NET_DEV_TYPE_PPP	PPP device

(4) Function of driver device

Device driver needs to support the below functions. These functions are called from appropriate protocol stack.

Prototype	Description	Requirement
ER dev_ini(UH dev_num)	Device initialization	Require
ER dev_cls(UH dev_num)	Device release	No require
ER dev_snd(UH dev_num, T_NET_BUF *pkt)	Send packet to network	Require
ER dev_ctl(UH dev_num, UH opt, VP val)	Device control	No require
ER dev_ref(UH dev_num, UH opt, VP val)	Device status acquisition	No require
void dev_cbk(UH dev_num, UH opt, VP val)	Notify event from device (callback function)	No require

(5) MAC address

Set unique value to specify hardware.

```
union {
    struct {
        UB      mac[6];   /* MAC address */
    }eth;
} cfg;
```

(6) Device header size

Header size of network device header size is set.

(7) Position writing network buffer

Network buffer data offset is set.

(8) Driver extension area

Device driver-specific parameter. Set to NULL (0) if unused.

3.2.2 Interface

dev_ini	Device initialization

【Format】

```
ER ercd = dev_ini(UH dev_num);
```

【Parameter】

UH	dev_num	Device number

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Device number is wrong
E_OBJ	Already initialized
E_PAR	Illegal value set in T_NET_DEV
<0	Other errors (implementation dependent)

【Explanation】

Initialize device. This function is called to initialize device from protocol stack. Before calling this function, it is necessary to register device information in T_NET_DEV.

dev_cls	Device release

【Format】

```
ER ercd = dev_cls(UH dev_num);
```

【Parameter】

UH	dev_num	Device number

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Device number is wrong
E_OBJ	Already released

【Explanation】

Release device.

dev_ctl Device control

【Format】

ER ercd = dev_ctl(UH dev_num, UH opt, VP val);

【Parameter】

UH	dev_num	Device number
UH	opt	Control code
VP	val	Value to be set

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Device number is wrong
E_PAR	Illegal parameter
E_OBJ	Already released

【Explanation】

The operation of this function is implementation dependent.

dev_ref Device status acquisition

【Format】

ER ercd = dev_ref(UH dev_num, UH opt, VP val);

【Parameter】

UH	dev_num	Device number
UH	opt	Status code
VP	val	Acquire value

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Device number is wrong
E_PAR	Illegal parameter
E_OBJ	Already released

【Explanation】

The operation of this function is implementation dependent.

dev_snd	Packet transmission

【Format】

```
ER ercd = dev_snd (UH dev_num, T_NET_BUF *pkt);
```

【Parameter】

UH	dev_num	Device number
T_NET_BUF	*pkt	Pointer to network buffer

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_WBLK	Packet is registered in queue (not error)
E_ID	Device number is wrong
E_PAR	Illegal parameter
E_TMOUT	Packet transmission timed out
E_OBJ	Device status was wrong already

【Explanation】

This function transmits packet to Ethernet.

An example of integration

```
ER dev_snd(UH dev_num, T_NET_BUF *pkt)
{
/* Copy to Ethernet frame (IP/TCP/UDP/Payload) */
memcpy(txframe,   pkt->hdr,   pkt->hdr_len);
   /* Transmit to network */
   xmit_frame(txframe);
   return E_OK;
}
```

In the above example, the process of protocol stack is blocked by device driver. The next example shows the example that using queue and no blocking.

Non-blocking example

```
ER dev_snd(UH dev_num, T_NET_BUF *pkt)
{
queue_tx(pkt);       /* register packet in queue */
return E_WBLK;   /* Non-blocking */
}

void queue_tx_task(void)
{
dequeue_tx(pkt); /* Removing packet from queue */
/* Copy to Ethernet frame (IP/TCP/UDP/Payload) */
memcpy(txframe,   pkt->hdr,   pkt->hdr_len);
xmit_frame(txframe);   /* Transmit to network */
if (transmission timeout) {
pkt->ercd = E_TMOUT;     /* Set time out */
}
net_buf_ret(pkt);
}
```

In dev_snd transmission process is not executed, packet will register in queue and return E_WBLK. Actual packet transmission process is executed by another task and release of network buffer is also executed there too.

dev_cbk	Device event notification

【Format】

```
void dev_cbk(UH dev_num, UH opt, VP val);
```

【Parameter】

UH	dev_num	Device number
UH	opt	Event code
UH	val	Event value

【Return value】

None

【Error code】

None

【Explanation】

This function is to notify an event to the application from device driver. This function is implementation dependent.

3.2.3 packet routing

To send a packet to the upper protocol stack from device driver, it uses the following API.
※This API cannot be used from Applications.

net_pkt_rcv	Sending packet to protocol stack

【Format】

```
void net_pkt_rcv(T_NET_BUF *pkt);
```

【Parameter】

T_NET_BUF	*pkt	Pointer to network buffer

【Return value】

None

【Error code】

None

【Explanation】

This function is to send packet to the upper protocol. The below example shows the example for sending packet to upper protocol stack from device driver.

Example

```
/* Network buffer allocation */
T_NET_BUF *pkt;
net_buf_get(&pkt, len, TMO);

/* Set received Ethernet header to network buffer */
pkt->hdr = pkt->buf + 2;
pkt->hdr_len = ETH_HDR_SZ;
memcpy(pkt->hdr, rx_frame, pkt->hdr_len);

/* Set received IP payload to network buffer */
pkt->dat= pkt->hdr + pkt->hdr_len;
pkt->dat_len = rx_frame_len – pkt->hdr_len;
memcpy(pkt->dat, rx_frame + pkt->hdr_len, pkt->dat_len);

/* Device information setting*/
pkt->dev = dev;

/* Transfer network buffer to protocol stack */
net_pkt_rcv(pkt);
```

Release of network buffer is executed by net_pkt_rcv(). net_pkt_rcv() must be called from task context.

3.2.4 Loopback interface

TCP/IP stack provides a loopback interface that wrap back packets in network device drivers. By using DDR_LOOPBACK_NET.c, packets sent from that interface are notified to the TCP/IP stack.

Unlike the general loopback interface (represented by 127.0.0.1), the TCP/IP stack sets static IP and MAC addresses. To use the loopback interface with the TCP/IP stack, select Loopback as the device type when registering the configurator interface.

To receive a packet from the loopback interface, the destination of the packet must be the assigned IP address. Address resolution using ARP is also performed when using the loopback interface.

3.2.5 T_NET_DEV information registration example

T_NET_DEV information registration example is showed as bellows.

T_NET_DEV information registration example

```
T_NET_DEV gNET_DEV[] = {
    {
        "lan0",              /* Device Name      */
        1,                   /* Device Number    */
        NET_DEV_TYPE_ETH,    /* Device Type      */
        0,                   /* Status           */
        0,                   /* Flags            */
        eth_ini,             /* Device Init      */
        eth_cls,             /* Device Close     */
        eth_ctl,             /* Device Configure */
        eth_sts,             /* Device Status    */
        eth_snd,             /* Device Transmit  */
        eth_cbk,             /* Device Callback  */
        0,
        {{{ 0x12, 0x34, 0x56, 0x78, 0x9A, 0xBC }}},   /* MAC Address */
        ETH_HDR_SZ,                                   /* Link Header Size */
        CFG_NET_BUF_OFFSET                            /* Network buffer data Offset */
    }
};
```

3.3 Memory management

In protocol stack, it uses network buffer in memory management. By using network buffer, it can allocate the empty block of memory actively. The following diagram shows an example of memory allocation. First, device driver which receives data from hardware will use API network buffer, then allocate memory (net_buf_get). Next, it will set necessary information in allocated memory and then send packet to the upper layer protocol stack (net_pkt_rcv).

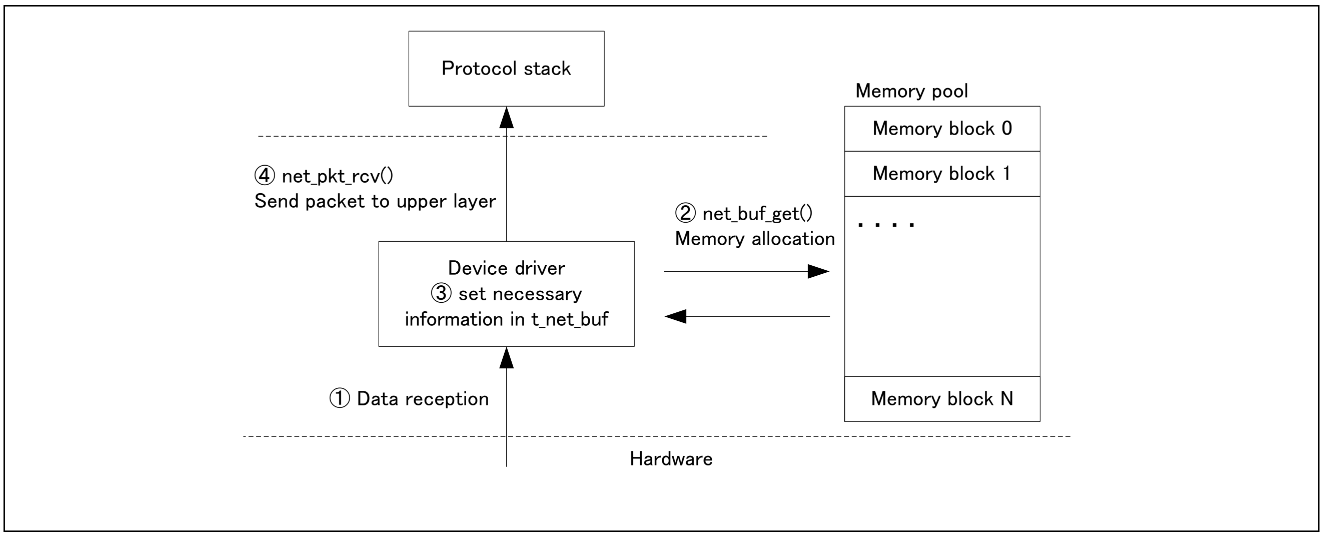


Fig. 3.8 Memory allocation diagram

3.3.1 Network buffer

In TCP/IP stack uses network buffers to send and receive packets. The network buffer is implemented using the memory allocator provided by the OS. The network buffer size and the number of network buffers can be defined by each application according to the communication volume and MTU size.

TCP/IP stack calls the net_memini (), net_memget (), net_memret (), net_memext () APIs for network buffer initialization, acquisition, release, and termination procedures. In addition, the maximum size of the network buffer that can be acquired must be set in CFG_NET_BUF_SZ in advance.

Network buffer organization (T_NET_BUF)

```
typedef struct t_net_buf {
        UW              *next;          /*Reserve */
        ID              mpfid;          /* ID memory pool */
        T_NET           *net;           /* Network interface */
        T_NET_DEV       *dev;           /* Network device */
        T_NET_SOC       *soc;           /* Socket */
        ER              ercd;           /* Error code */
        UH              flg;            /* Flag used to control protocol stack */
        UH              seq;            /* Fragment sequence*/
        UH              dat_len;        /* Data size of packet */
        UH              hdr_len;        /* Header size of packet */
        UB              *dat;           /* Showing data position in packet (buf) */
        UB              *hdr;           /* Showing header position in packet (buf) */
        UB              buf[];          /* Actual packet*/
} T_NET_BUF ;
```

The TCP/IP stack uses T_NET_BUF to transceive packet between protocol and device driver or the protocols.
In TCP/IP, the actual packet data are stored in ‘buf’,‘*dat’,‘*hdr’, ‘hdr_len’,‘dat_len’ are used to access to that.

(1) The access between protocol stack and device driver by network buffer

Member	Send (protocol stack->driver)	Receive (driver->protocol stack)
dev	This is set for &gNET_DEV[dev_num-1]. dev_num is specified by application.	This is set for &gNET_DEV[eth_dev_num-1] eth_dev_num is set at initialization.
ercd	This shows the reason why driver fails to send. If driver succeed to send, this remains the size specified by protocol stack.	Not used.
flg	This controls whether hardware checksum is enabled or disabled; - HW_CS_TX_IPH4(0x0040) (IP header checksum) - HW_CS_TX_DATA(0x0080) (payload data checksum)	This controls whether hardware checksum is enabled or disabled; - HW_CS_TX_IPH4(0x0040) (IP header checksum) - HW_CS_RX_DATA(0x0020) (payload data checksum) If above checksum result has an error, please set below bits: - HW_CS_IPH4_ERR(0x0100) (IP header checksum error) - HW_CS_DATA_ERR(0x0200) (payload data checksum error)
hdr/hdr_len	This is set the head address and size of frame data which protocol stack sends. Driver sends data addressed between hdr and hdr_len offset.	This Is set the head address and size of frame data which driver received data. (In the case of Ethernet frame, header size is 14 Bytes)
dat/dat_len	Not used.	This is set head address of data and data size following the frame header received by driver. (In the case of Ethernet, dat shows the position shifted from hdr to hdr_len offset.)
buf[]	Packet This stores actual packet data. buf[0] and buf[1] are 4Byte alignments control buffer. The data to send or receive can be written upper buf[2].	

3.3.2 API network buffer

※This API network buffer cannot be used from application.

net_buf_get	Network buffer allocation

【Format】

ER ercd = net_buf_get(T_NET_BUF **buf, UH len, TMO tmo);

【Parameter】

T_NET_BUF	**buf	Address of buffer that allocate memory
UH	len	Number of allocating bytes
UH	tmo	Timeout specification

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Set wrong parameter value
E_NOMEM	Unable to allocate memory
E_TMOUT	Timeout

【Explanation】

Allocate memory from memory pool. Allocated buffer address returns to buf.

net_buf_ret	Network buffer release

【Format】

void net_buf_ret(T_NET_BUF *buf);

【Parameter】

T_NET_BUF	*buf	Address of buffer that free memory

【Return value】

None

【Error code】

None

【Explanation】

Give back memory to the memory pool. If the socket is associated with network buffer, notify the free memory event to the socket.

3.4 Memory processing I / O

Comparison and writing process of contiguous memory occurred in the protocol are able to defined by user, so that it does not depend on device or compilation environment.

For devices with features DMA, Processing memory copy can use the DMA transfer instead of the memcpy() of standard library.

3.4.1 Memory processing I / O

※memory I/O API must be defined in application always.

net_memset	Fill block of memory

【Format】

void* net_memset(void* d, int c, SIZE n);

【Parameter】		
void*	d	Pointer to the block of memory to fill
int	c	Value to be set
SIZE	n	Number of bytes to be set

【Return value】		
void*	d	Pointer to the block of memory to fill

【Explanation】

If the memory settings are successful, please return the destination pointer that is specified in the argument.

net_memcpy	Copy bytes in memory

【Format】

void* net_memcpy(void* d, const void* s, SIZE n);

【Parameter】		
void*	d	Pointer to the destination of memory
const void*	s	Pointer to the source of data
SIZE	n	Number of bytes to copy

【Return value】		
void*	d	Pointer to the destination of memory

【Explanation】

If the memory copies are successful, please return the destination pointer that is specified in the argument.

net_memcmp Compare two blocks of memory

【Format】

```
int net_memcmp(const void* d, const void* s, SIZE n);
```

【Parameter】

const void*	d	pointer to blocks of memory1
const void*	s	pointer to blocks of memory2
SIZE	n	Number of bytes to compare

【Return value】

int	Comparison result

【Explanation】

Please return 0 if the same value in the specified number of bytes. Otherwise, please return the non-zero.

3.5 Ethernet device driver

This device driver is for Ethernet MAC or Ethernet Switch included in R-IN32. Ethernet device driver is used by calling from network device driver.

This device driver functions are showed below.

- PHY mode set/get
- PHY speed set/get
- Receive frame filtering
- Dynamic configuration for multicast address filter
- Raw data send/receive API
- Direct MAC mode / Ether Switch mode
- Blocking transmit / non-blocking transmit
- VLAN

3.5.1 Ethernet device driver structure

Ethernet device driver operates with PHY driver which controls MDIO interface. This driver structure is showed below.

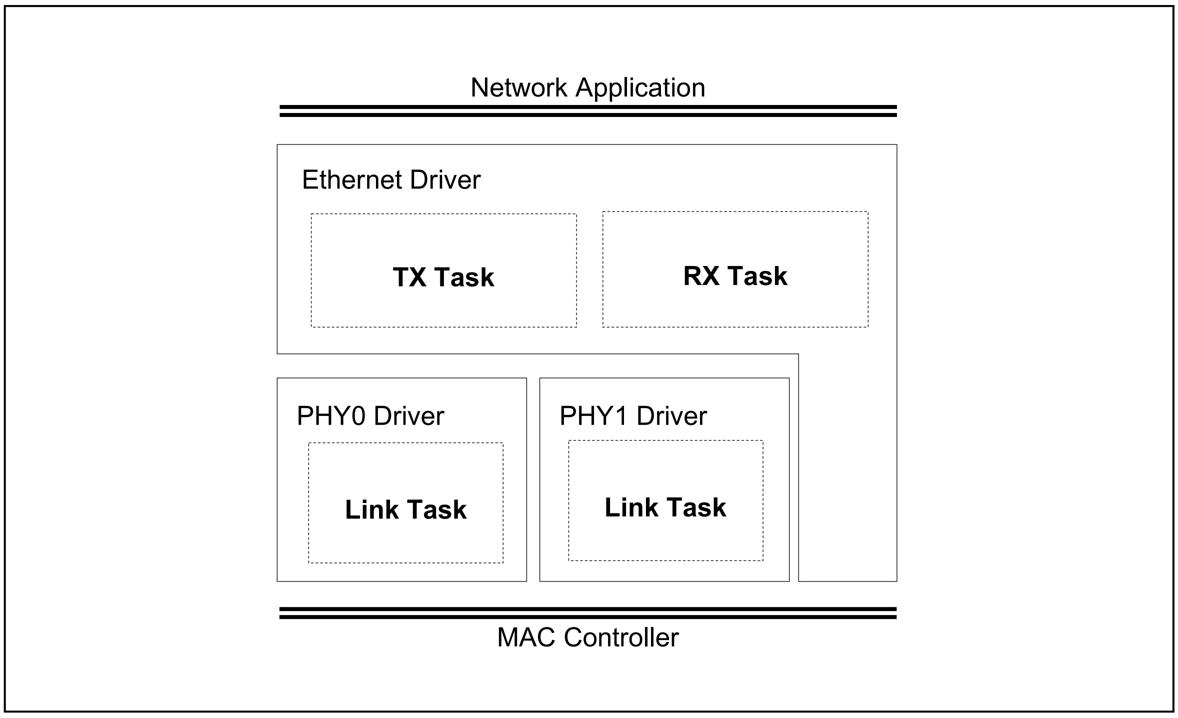


Fig.3.9 Ethernet device driver structure

3.5.2 Ethernet device driver API

3.5.2.1 Ethernet driver initialization - eth_ini()

【Prototype】

ER eth_ini(UH dev_num)

【Operation】

Initialize Ethernet driver

【Parameter】

UH	dev_num	Device number (1)

【Return value】

ER	E_OK	Success initialization
	E_ID	Undefined device number
	E_PAR	Invalid device number
	Others	Task wake up error, PHY driver initialization error

【Explanation】

This API initialize PHY and MAC controller, and wake up the task which controls Ethernet driver. This function must be called before Ethernet driver is used.

3.5.2.2 Ethernet driver closed - eth_cls()

【Prototype】

ER eth_cls(UH dev_num)

【Operation】

Ethernet driver is closed

【Parameter】

UH	dev_num	Device number (1)

【Return value】

ER	E_OK	Exit without error

【Explanation】

PHY, MAC controller stops.

3.5.2.3 Ethernet frame transmit - eth_snd()

【Prototype】

ER eth_snd(UH dev_num, T_NET_BUF *pkt)

【Operation】

Ethernet frame transmit

【Parameter】

UH	dev_num	Device number(1)
UH	pkt->hdr	Address for transmit data
UB*	pkt->hdr_len	Transmit data length

【Return value】

ER	E_OK	Success for transmit
	E_TMOUT	Link down
	E_WBLK	Non-blocking reception (Only when non-blocking transit)
	E_NOMEM	No enough memory
	E_SYS	MAC controller error

【Explanation】

When blocking transmit, this function returns after waits the interrupt for transmit complete. When non-blocking transmit, this function returns with E_WBLK after transmit success. If E_WBLK is returned, Ethernet driver calls back “eth_raw_snddone()” function when Ethernet frame transmit completed.

3.5.2.4 Ethernet frame transmit end report - eth_raw_snddone()

【Prototype】

void eth_raw_snddone(T_NET_BUF *pkt)

【Operation】

Ethernet frame transmit complete call-back function

【Parameter】

T_NET_BUF*	pkt	pkt specified to eth_snd()
ER	pkt->ercd	Transmit result

【Return value】

void

【Explanation】

Ethernet driver is called when non-blocking transmit is completed. This function needs to be registered by application before it’s used. How to registration is showed below.

```
/* Function body of transmit complete note */
void eth_raw_snddone(T_NET_BUF *pkt)
{
}

/* Registration to transmit completion report function */
eth_ctl(1, ETH_OPT_RAW_SNDDONE, (VP)eth_raw_snddone);
```

The TX task must not be stopped, because this function operates under TX task.

3.5.2.5 Ethernet frame reception report - eth_raw_rcv()

【Prototype】

void eth_raw_rcv(VP fram, UH len)

【Operation】

Ethernet frame reception report function

【Parameter】

VP	fram	Reception data address
UH	len	Reception data length

【Return value】

void

【Explanation】

When Ethernet driver receive a frame, this function is called. This function needs to be registered to application before it is used. Registration example shows below.

```
/* Receive report function */
void eth_raw_rcv(VP p, UH len)
{
}

/* Receive report function registration */
eth_ctl(1, ETH_OPT_RAW_RXFNC, (VP)eth_raw_rcv);
```

Rx task must not be stopped because this function operates Rx task. If Ethernet driver is shared with TCP/IP stack, this function is called in driver earlier than in TCP/IP stack. Therefore, received frame data cannot be modified.

3.5.2.6 PHY ability setting - set_phy_mode()

【Prototype】

ER set_phy_mode(PHY_MODE *mode)

【Operation】

PHY ability setting

【Parameter】

PHY_MODE*	mode	PHY ability
UW	mode->mode	Speed / Duplex
UB	mode->nego	Auto-negotiation on(TRUE)/off(FALSE)
UB	mode->ch	Target PHY channel

【Return value】

ER	E_OK	Success
	E_PAR	Invalid parameter
	others	PHY driver configuration error

【Explanation】

Application can configure PHY ability anytime. The configuration of speed and duplex is selected from Table. 3.1. Target PHY is configured to ETH_INT_MII_PHY0 or ETH_INT_MII_PHY1.

```
struct PHY_MODE pmod;

pmod.mode = LAN_AUTO_ABILITY;
pmod.nego = TRUE;
pmod.ch = ETH_INT_MII_PHY0;

set_phy_mode(&pmod);
```

Table. 3.1 PHY ability setting

Value	Description
LAN_10T_HD	10M/Half-duplex
LAN_10T_FD	10M/Full-duplex
LAN_100TX_HD	100M/Half-duplex
LAN_100TX_FD	100M/Full-duplex
LAN_1000T_HD	1G/Half-duplex
LAN_1000T_FD	1G /Full-duplex
LAN_AUTO_ABILITY	Auto select

3.5.2.7 Get PHY ability - get_phy_mode()

【Prototype】

ER get_phy_mode(PHY_MODE *mode)

【Operation】

Get PHY ability

【Parameter】

PHY_MODE*	mode	PHY ability

【Return value】

ER	E_OK	Success
	E_PAR	Invalid parameter
	others	PHY driver ability get error

【Explanation】

This function is used to get PHY ability (speed, duplex, auto-negotiation on/off), and link status. Application can get PHY ability anytime. Example is showed below.

```
struct PHY_MODE pmod = {0};

/* Target PHY is set by ETH_INT_MII_PHY0 or ETH_INT_MII_PHY1 */
pmod.ch = ETH_INT_MII_PHY0;
get_phy_mode(&pmod);
```

pmod.mode shows PHY ability (reference Table. 3.1).
pmod.nego shows auto-negotiation enabled(TRUE) or disabled(FALSE).
pmod.link shows link-up(TRUE) or link-down(FALSE).

If a PHY is link-down, please notice that PHY ability shows invalid value.

3.5.2.8 Multicast address filter mode setting - set_mcast_filter_mode()

【Prototype】

ER set_mcast_filter_mode(UINT mode)

【Operation】

Set multicast address filter mode

【Parameter】

UINT	mode	Filter mode

【Return value】

ER	E_OK	Success
	E_PAR	Invalid parameter

【Explanation】

Filter mode has three modes showed below.

Value	Mode
MCRX_MODE_ALLOW	All multicast address receive mode
MCRX_MODE_DENY	Denied to receive multicast address mode
MCRX_MODE_FILTER	Specified multicast address receive mode

Caution 1 When independent address is set, choose MCRX_MODE_FILTER.

Caution 2 After MCRX_MODE_DENY is set, already registered multicast address is deleted all.

3.5.2.9 Add multicast address filter - add_mcast_filter()

【Prototype】

ER add_mcast_filter(MAC_FILTER *adr)

【Operation】

Add multicast address to receive

【Parameter】

MAC_FILTER*	adr	Multicast address to receive
UB	adr->mac[6]	Multicast address to add
UB	adr->bitnum	Valid bit number (0, 40-48)

【Return value】

ER	E_OK	Success
	E_PAR	Invalid parameter

【Explanation】

Filter mode must be set to MCRX_MODE_FILTER to add multicast address for receive.

Multicast address is shown as combination with lower 23 bits of class D IP address (1st octets is 0xE0~0xEF(224~239)) and upper 25 bits (0x01.0x00.0x5e.0x00/25). Application has to calculate receive MAC address from entering multicast IP address. Valid bits are specified 40 bits (5 octets) to 48 bits bit-by-bit. If 0 is set, it seems to be no valid bits (equal 48).

In case that all MAC address 01:00:5e:00:01:* (* is any value) can be received, the example is showed below.

```
MAC_FILTER adr;
UB macadr[] = {0x01, 0x00, 0x5e, 0x00, 0x01, 0x00};

memcpy(&adr.mac[0], macadr, 6);
adr.bitnum = 40;
add_mcast_filter(&adr);
```

Note: Please add the multicast address (224.0.0.1) which targeget all nodes. If this address is not added, receive operation can’t work.

3.5.2.10 Call back event from Ethernet driver

【Prototype】

void eth_cbk(UH dev_num, UH opt, VP val)

【Operation】

Call back event from Ethernet driver

【Parameter】

UH	dev_num	Device number
UH	opt	Event type
VP	val	Event description

【Return value】

void

【Explanation】

This function is defined by network application. It is possible to control operation for asynchronous event or specific application operation by registering call back function to device configuration (T_NET_DEV gNET_DEV[]).

Event type and contents are showed below.

Event type	Event contents	Factor
EV_CBK_DEV_INIT	Always 0	When Ethernet driver is initialized
EV_CBK_DEV_LINK	Link down:0, Link up:1	When link status is changed

Caution Currently, link event cannot be detected.

3.5.2.11 Ethernet driver option setting - eth_ctl()

【Prototype】

ER eth_ctl(UH dev_num, UH opt, VP val)

【Operation】

Ethernet driver option setting

【Parameter】

UH	dev_num	Device number
UH	opt	Option type
VP	val	Setting value

【Return value】

ER	E_OK	Success
	E_PAR	Invalid parameter

【Explanation】

This function calls Ethernet driver or PHY driver setting function. Available options are showed below.

Option type	Description	Available value
ETH_OPT_PHY_MODE	PHY ability	same as set_phy_mode()
ETH_OPT_MCRX_MODE	Multicast address filter mode setting	same as set_mcast_filter_mode()
ETH_OPT_MCRX_ADR	Add multicast address filter	same as add_mcast_filter()
ETH_OPT_RAW_RXFNC	Ethernet frame receive report	same as eth_raw_rcv()
ETH_OPT_RAW_SNDDONE	Frame tranmit completaion report	same as eth_raw_snddone()

3.5.2.12 Get Ethernet driver option - eth_sts()

【Prototype】

ER eth_sts(UH dev_num, UH opt, VP val)

【Operation】

Get Ethernet driver option

【Parameter】

UH	dev_num	Device number
UH	opt	Option type
VP	val	Option contents

【Return value】

ER	E_OK	Success
	E_PAR	Invalid value

【Explanation】

This function calls Ethernet driver and PHY driver functions. Available option is below.

Option type	Option contents	Available value
ETH_OPT_PHY_MODE	Get PHY ability	same as get_phy_mode()

3.5.3 Configuration

Following configuration is written to DDR_ETH_CFG.h depends on purpose.

#define PHY_ADR0

LAN1 PHY address for R-IN32M4-CL3

#define PHY_ADR1

LAN2 PHY address for R-IN32M4-CL3

#define PROMISCUOUS_FILTER_MODE

Promiscuous filter mode. (0: receive all frames, 1: receive only self-station frame)

#define ETH_EARLY_TX_ENA

Early transmit mode. (0: disabled, 1: enabled)

If early transmit mode is enabled, it seems finish transmitting as soon as data was transmitted to FIFO in MAC controller.

#define ETH_TX_ASYNC

Non-blocking transmit mode. (0: disabled, 1: enabled)

If non-blocking mode is enabled, transmission is operated by send task of Ethernet drive.

#define USE_ETHSW

Ethernet switch is used or not. (0: not used, 1: used)

#define USE_ETHSW_MGTAG

Ethernet switch management tag is used or not. (0: not used, 1: used)

3.5.4 Cautions regarding Ethernet device driver

3.5.4.1 TCP/UDP hardware checksum function

Ethernet device driver uses hardware checksum function of R-IN32. Therefore, checksum calculation is not operated inside protocol stack.

T_NET_BUF structure member “flg” and global variable gNet[device number-1].flag can control whether checksum operation is used or not.

Following ensamples show how hardware checksum function is enabled.

Enables receive hardware checksum

```
T_NET_BUF *pkt;

pkt->flg   |= (HW_CS_RX_IPH4 | HW_CS_RX_DATA);
```

Enables transmit hardware checksum

```
T_NET      *net;

net = &gNET[eth_devnum-1];

/* set hardware checksum flag */
net->flag |= (HW_CS_TX_IPH4 | HW_CS_TX_DATA);
```

3.6 PHY driver

Ethernet driver and PHY driver I/F are defined to global variable PHY_IO gPHY_IO[]. gPHY_IO[] is arrays to define some PHYs.

PHY driver API registered to this variable is called from Ethernet driver. Last element of gPHY_IO[] arrays must be set 0 to all of PHY_IO member and terminated. PHY_IO member is described below.

```
typedef struct phy_io {
    UW phy_id;
    UW phy_adr;
    ER (*phy_ini)(ID flg, UW id, UW adr);
    ER (*phy_ext)(void);
    ER (*phy_set_mode)(UW mode, UB nego);
    ER (*phy_get_mode)(UW *mode, UB *nego, UB *link);
}PHY_IO;
```

Member	Description
phy_id	PHY ID (ID number of LAN1~LAN4. If target is LAN1, this member is 1)
phy_adr	PHY address
phy_ini	Initialize function
phy_ext	Exit function
phy_set_mode	PHY ability setting function
phy_get_mode	PHY ability getting function

3.6.1 PHY driver API

3.6.1.1 Initialize PHY driver - phy_ini()

【Prototype】

ER phy_ini(ID flg, UW id, UW adr)

【Operation】

Initialize PHY driver

【Parameter】

ID	flg	Event flag ID
UW	id	PHY ID
UW	adr	PHY address

【Return value】

ER	E_OK	Success
	Others	Error when task start

【Explanation】

This function is called from Ethernet driver when eth_ini() function is executed. Event flag ID gives Ethernet driver link event. PHY ID gives event pattern when link event is noticed.

3.6.1.2 Exit PHY driver - phy_ext()

【Prototype】

ER phy_ext(void)

【Operation】

Exit PHY driver

【Parameter】

void

【Return value】

ER	E_OK	Success
	Others	Error when task finish

【Explanation】

This function is called from Ethernet driver when eth_cls() function is executed. This driver terminates link task.

3.6.1.3 PHY ability setting - phy_set_mode()

【Prototype】

ER phy_set_mode(UW mode, UB nego)

【Operation】

PHY ability setting

【Parameter】

UW	mode	Speed, duplex
UB	nego	Auto negotiation enabled(TRUE) / disabled(FALSE)

【Return value】

ER	E_OK	Finished

【Explanation】

This function is called from Ethernet driver when set_phy_mode() function is executed. Speed and duplex has the value referred from Table. 3.2. This value is different from Ethernet driver API set_phy_mode() parameter speed and duplex. Default value is PHY_AUTO_ABILITY when PHY driver has been initialized.

Table. 3.2 PHY ability setting

Settings	Description
PHY_10T_HD	10M/Half-duplex
PHY_10T_FD	10M/Full-duplex
PHY_100TX_HD	100M/Half-duplex
PHY_100TX_FD	100M/Full-duplex
PHY_1000T_HD	1G/Half-duplex
PHY_1000T_FD	1G /Full-duplex
PHY_AUTO_ABILITY	Auto selected

3.6.1.4 Get PHY ability - phy_get_mode()

【Prototype】

ER phy_get_mode(UW *mode, UB *nego, UB *link)

【Operation】

Get PHY ability

【Parameter】

UW*	mode	Speed and duplex
UB*	nego	Auto negotiation enabled / disabled
UB*	link	Link up / down

【Return value】

ER	E_OK	Finished

【Explanation】

This function is called from Ethernet driver when get_phy_mode() function is executed. Speed and duplex has the value referred from Table. 3.2. This value is different from Ethernet driver API set_phy_mode() parameter speed and duplex.

3.6.2 Link event notification

Rx task in Ethernet driver waits for receiving frame or changing link event state. The event shows link state change is set (set_flg()) by link task in PHY driver or interrupt handler.

The event flag ID (1st argument) in set_flg(id, ptn) has specified event flag ID when PHY driver starts. The event bit pattern (2nd argument) has logical OR "PHY_LINK_EVT" and PHY ID. PHY_LINK_EVT means link event, PHY ID is specified when PHY driver starts.

4. Network configuration

It’s explained to configure TCP/IP protocol stack in this chapter.

4.1 Configuration of TCP/IP stack

TCP/IP stack parameters such as IP address and transmission buffer size can be configured by editing net_cfg.c.

4.1.1 Configuration list

The following is configurable parameter list.

The application can modify macro values defined as “#define CFG_XXX” (XXX is any), and cannot modify other macros or variables directly. Almost initial values are defined by “DEF_XXX” macro.

Configuration content [Unit]	Definition	Initial value	Minimum value	Maximum value
The number of data link devices	CFG_NET_DEV_MAX	1	1	2
The maximum number of all protocol sockets	CFG_NET_SOC_MAX	10	1	1000
The maximum number of TCP sockets (*1) (*3)	CFG_NET_TCP_MAX	5	0	1000
The number of ARP entries	CFG_NET_ARP_MAX	8	1	32
The number of multicast entries	CFG_NET_MGR_MAX	8	1	100
The number of IP reassembly queues	CFG_NET_IPR_MAX	2	1	16
Maximum network buffer size [Byte] (*2)	CFG_NET_BUF_SZ	8192	2048	8192
The number of network buffers	CFG_NET_BUF_CNT	8	2	31
Network buffer data offset (not allowed to change) (*2)	CFG_NET_BUF_OFFSET	2	42	42
MTU size (*2) (*4)	CFG_PATH_MTU	1500	576	1500
The number of ARP retry (*4)	CFG_ARP_RET_CNT	3	0	10
Timeout value for ARP retry [ms] (*4)	CFG_ARP_RET_TMO	1000	500	10000
Clear timeout of ARP cache [ms] (*4)	CFG_ARP_CLR_TMO	1200000	1000	3600000
TTL value of IP header (*4)	CFG_IP4_TTL	64	1	255
TOS value of IP header (*4)	CFG_IP4_TOS	0	0	255
Waiting time IP fragment packet [ms] (*4)	CFG_IP4_IPR_TMO	10000	100	60000
TTL value of multicast IP header (*4)	CFG_IP4_MCAST_TTL	1	1	255
Timeout value of IGMPv1 [ms] (*4)	CFG_IGMP_V1_TMO	40000	40000	120000
Timeout value of IGMP report [ms] (*4)	CFG_IGMP_REP_TMO	10000	10000	30000
MSS(TCP/IPv4) (MTU-IP header-TCP header) (*4)	CFG_TCP_MSS	1460	536	1460
TCP/RTO (retry timeout) initial value [ms] (*4)	CFG_TCP_RTO_INI	3000	2000	3000
TCP/RTO (retry timeout) minimum value [ms] (*4)	CFG_TCP_RTO_MIN	500	200	500
TCP/RTO (retry timeout) maximum value [ms] (*4)	CFG_TCP_RTO_MAX	60000	30000	60000
TCP transmission buffer size [Byte] (*3) (*4) (*5)	CFG_TCP_SND_WND	1024	1024	8192
TCP reception buffer size [Byte] (*4) (*5)	CFG_TCP_RCV_WND	1024	1024	8192
Duplicate ACK number of retry beginning [reception count] (*4)	CFG_TCP_DUP_CNT	4	1	10

Configuration content [Unit]	Definition	Initial value	Minimum value	Maximum value
TCP/SYN transmission timeout [ms] (*4)	CFG_TCP_CON_TMO	75000	10000	75000
TCP/data transmission timeout [ms] (*4)	CFG_TCP_SND_TMO	64000	10000	64000
TCP/FIN transmission timeout [ms]	CFG_TCP_CLS_TMO	75000	10000	75000
TCP/2MSL timeout [ms] (*4) (unused)	CFG_TCP_CLW_TMO	20000	0	20000
TCP/delay ACK transmission period [ms] (*4)	CFG_TCP_ACK_TMO	200	200	1000
TCP/Keep-Alive notification count (in case of 0 Keep-Alive is disabled)	CFG_TCP_KPA_CNT	0	0	100
TCP/Keep-Alive notification period [ms]	CFG_TCP_KPA_INT	1000	1000	60000
Non-communication time until TCP/Keep-Alive starts [ms]	CFG_TCP_KPA_TMO	7200000	10000	14400000
The queuing number of reception packet (*4)	CFG_PKT_RCV_QUE	1	1	10
Received sequence guaranteed queuing number	CFG_TCP_RCV_OSQ_MAX	6	-	-
Received packet checksum verification invalid flag	CFG_PKT_CTL_FLG	0	-	-
Waiting time ARP PROBE packet transmission [ms]	CFG_ARP_PRB_WAI	1000	1000	3000
The number of ARP PROBE packet transmission	CFG_ARP_PRB_NUM	3	1	6
ARP PROBE packet transmission period (minimum) [ms]	CFG_ARP_PRB_MIN	1000	100	1000
ARP PROBE packet transmission period (maximum) [ms]	CFG_ARP_PRB_MAX	2000	200	2000
Waiting time ARP ANNOUNCE packet [ms]	CFG_ARP_ANC_WAI	2000	200	2000
The number of ARP ANNOUNCE packet transmission	CFG_ARP_ANC_NUM	2	1	4
ARP ANNOUNCE packet transmission period [ms]	CFG_ARP_ANC_INT	2000	200	2000

In the setting item whose unit is [millisecond], it is reflected as an approximate value depending on the accuracy of the timer used.

*1: Must be less than or equal to CFG_NET_SOC_MAX. Also, the difference from CFG_NET_SOC_MAX is the maximum number of non-TCP sockets.

*2: The size of the network buffer must be larger than the total size of the network buffer management structure size (44Byte), MTU, data link header size (14Byte for Ethernet), and network buffer data write position (default 2Byte).

*3: The TCP send buffer uses the global variable UB gTCP_SND_BUF [] in common regardless of the device used. gTCP_SND_BUF [] determines this size by CFG_TCP_SND_WND × CFG_NET_TCP_MAX. If you use multiple devices with different TCP send buffer sizes, you need to set gTCP_SND_BUF [] according to the maximum value.

*4: It can be set for each device to be used. Set device number -1 as an index in gNET_CFG [].

*5: The buffer size ranges from 4 bytes to 32 kilobytes and is specified as a power of 2 (1024, 2048, 4096, and so on).

4.1.2 IP address

Set up IP address. Because every network device needs an IP address, please register CFG_NET_DEV_MAX part for IP address.

The following is the example for set up of IP address:192.168.1.10, gateway : 192.168.1.1, subnet mask: 255.255.255.0.

```
T_NET_ADR gNET_ADR[] = {
  {  /*   for Device 1   */
     0x0,              /* Must be 0 */
     0x0,              /* Must be 0 */
     0xC0A8000A,  /* Setting IP address 192.168.1.10     */
     0xC0A80001,  /* Gateway 192.168.1.1       */
     0xFFFFFF00,  /* Subnet mask 255.255.255.0 */
  }
};
```

4.1.3 Device Driver

Set device driver. Please register CFG_NET_DEV_MAX part for device driver.

```
T_NET_DEV gNET_DEV[]   =   {
   { ..}
}
```

Please refer to 3.2 Network device driver for more details.

4.1.4 Information table of protocol stack

Set global variable of protocol stack as below.

```
const VP net_inftbl[] = {
0,                              /* Necessarily specify 0*/
(VP)gNET_SOC,          /* Set NULL in case of not using socket */
(VP)gNET_TCP,          /* Set NULL in case of not using socket*/
(VP)gNET_IPR,          /* Set NULL in case of not using IP reassembly function */
(VP)gNET_MGR,          /* Set NULL in case of not using IGMP*/
(VP)gTCP_SND_BUF,   /* Set NULLin case of not using TCP socket*/
};
```

4.1.5 Network information management resources

In the TCP/IP stack, in order to manage network information, it is necessary to prepare an information management area according to the number of devices and sockets with wide area variables as shown below.

T_NET_STS_DEV	net_cfg_sts_dev[CFG_NET_DEV_MAX]	Device information
T_NET_STS_IFS	net_cfg_sts_ifs[CFG_NET_DEV_MAX]	TCP/IP information
T_NET_STS_IFS	net_cfg_sts_ifs_tmp[CFG_NET_DEV_MAX]	TCP/IP information (Temporary)
T_NET_STS_ARP	net_cfg_sts_arp[CFG_NET_ARP_MAX]	ARP information
T_NET_STS_SOC	net_cfg_sts_soc[CFG_NET_SOC_MAX]	Socket information
VP	net_cfg_sts_ptr[4 + CFG_NET_DEV_MAX]	Status pointer
VP	net_cfg_sts_ptr_tmp[4 + CFG_NET_DEV_MAX]	Status pointer

T_NET_STS_CFG	T_NET_STS_CFG gNET_STS_CFG = {
	    net_cfg_sts_dev,
	    net_cfg_sts_ifs,
	    net_cfg_sts_ifs_tmp,
	    net_cfg_sts_arp,
	    net_cfg_sts_soc,
	    net_cfg_sts_ptr,
	    net_cfg_sts_ptr_tmp,
	    0
	};
		Network information management table

5. Description of application programming interface

5.1 Initialization of protocol stack

To use TCP/IP protocol stack, the initialization of protocol stack and initialization of network device are needed. Basically initialization is as follows:

```
Example of initialization code
/* Initialization of protocol stack */
        ercd    =     net_ini();
if (ercd != E_OK) {
          return ercd;
        }
/* Network device initialization (device number N) */
        ercd    =     net_dev_ini(N);
If (ercd != E_OK) {
return ercd;
}
```

5.2 Network Interface API

net_ini	Initialization of TCP/IP protocol stack

【Format】

ER ercd = net_ini(void);

【Parameter】

None

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

<0	Initialization failure

【Explanation】

Initialize the resource to be used by protocol stack. Kernel objects (tasks, memory pools, semaphores) to be used by protocol stack are also created and initialized simultaneously. Besides, the initial value is set in global variable used in protocol stack.

In case of using protocol stack, it needs to issue this API prior to any other API.

net_cfg Parameter setting of network interface

【Format】

```
ER ercd = net_cfg(UH num, UH opt, VP val);
```

【Parameter】

UH	num	Device number
UH	opt	Parameter code
VP	val	Value to set

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_NOSPT	Wrong parameter code
E_ID	Wrong parameter code
E_NOMEM	Too many multicast table

【Explanation】

Set up forIP address and subnet mask, broadcast address, multicast and others.

Setting sample
net_cfg(1, NET_BCAST_RCV, (VP)1); /* enable broadcast reception */

Parameter code	Data type	Meaning
NET_IP4_CFG	T_NET_ADR	Set IP address, subnet mask, gateway. Please hand pointer of T_NET_ADR to val.
NET_IP4_TTL	UB	Set TTL (Time to Live) Default is set 64.
NET_BCAST_RCV	UB	Set whether to receive broadcast or not. Set 1 to receive and 0 to not receive.
NET_MCAST_JOIN	UW	Register IP address of multicast group to join
NET_MCAST_DROP	UW	Set IP address of multicast group to drop
NET_MCAST_TTL	UB	Set TTL to be used in multicast transmission
NET_ACD_CBK	Callback function pointer	In this field, set the callback function to notify that it has detected an IP address conflict during operation. Notification feature is enabled by this setting of conflict detection.

net_ref Reference parameter for network interface

【Format】

```
ER ercd = net_ref(UH num, UH opt, VP val);
```

【Parameter】

UH	num	Device number
UH	opt	Parameter code
VP	val	Pointer to buffer of the value to get

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_NOSPT	Wrong parameter code
E_ID	Wrong device number

【Explanation】

Verify the basic setting of IP address and subnet mask, broadcast address and others.

Setting sample

```
UB bcast;
net_ref(1, NET_BCAST_RCV, (VP)&bcast);   /* reception status of broadcast */
```

Parameter code	Data type	Meaning
NET_IP4_CFG	T_NET_ADR	Get IP address, subnet mask, gate-way. Please hand pointer of T_NET_ADR to val
NET_IP4_TTL	UB	Get TTL (Time to Live).
NET_BCAST_RCV	UB	Get the receive status of broadcast
NET_MCAST_TTL	UB	Get TTL broadcast transmission

net_acd Detection IP Address Confliction

【API】

```
ER ercd = net_acd(UH dev_num, T_NET_ACD *acd);
```

【Parameter】

UH	num	Deviec Number
T_NET_ACD	*acd	Address Conflict Information

【Return Value】

ER	ercd	Success (E_OK) or Error Code

【Error Code】

E_ID	Illegal Device Number
E_PAR	Illegal Parameter
E_OBJ	Call Duplicate or Call host IP undefined
E_TMOUT	Time out ARP transmit
E_SYS	Detect IP address conflict
E_OK	No Detect IP address conflict

【DESCRIPTION】

This API is done in the device specified by dev_num, IP address conflict detection.

If it detects a conflict for the IP address, MAC address of the other party is stored in the conflict information of the argument.

If you want to detect conflicts in asynchronous IP address separately, you will need to register a callback function () API net_cfg with this API.

Note : Function is recommended that a maximum of about 10 seconds, call a dedicated task to attempt the detection of address conflicts.

acd_cbk	Callback IP address conflict detected.

【API】

```
ER acd_cbk(T_NET_ACD* acd);
```

【Parameter】

T_NET_ACD	*acd	Address Conflict Information

【Return Value】

ER	ercd	Success (E_OK) or Error Code

【DESCRIPTION】

This function is called when it detects an IP address conflict during operation. The conflict is stored what conflict information of the argument MAC address of the host that.

If the IP address for the conflict, continue to use the IP address in the host itself, please return the E_OK. Otherwise, please return the E_SYS.

Callback function is called on the task that received the ARP packet (the task that receives the Ethernet driver). Therefore, the callback function should be terminated immediately. Also, the callback function is not called while the IP address is being detected (net_acd () is being executed).

Use Cases

```
#define     ID_DEVNUM_ETHER   1     /* Device number */
/* Callback function at the time of detecting address conflicts */
ER acd_detect(T_NET_ACD * acd)
{
    return E_OK;
}

/* Network Initialize Function */
ER net_setup(void)
{
    ER ercd;
    T_NET_ACD acd;

    ercd = net_ini();
    if (ercd != E_OK) {
        return ercd;
    }
    ercd = net_dev_ini(ID_DEVNUM_ETHER);
    if (ercd != E_OK) {
        return ercd;
    }

    /* Detect IP Address Conflict */
    ercd = net_acd(ID_DEVNUM_ETHER, &acd);
    if (ercd == E_OK) {
        /* No Information IP Address conflict */
        /* Callback function at the time of detecting IP address conflicts */
        net_cfg(ID_DEVNUM_ETHER, NET_ACD_CBK, (VP)acd_detect);
    }
    else if (ercd == E_SYS) {
        /* MAC address is conflict a host of acd.mac IP address */
    } else {
        /* Failed to detect IP address conflict */
    }
    return ercd;
}
```

5.3 Network Device Control API

The Network Device Control API provides interface to access unifiedly from application to device driver. For each device, it specifies a device number to access this API. Device number is the specific number to identify the device.

net_dev_ini	Network device initialization

【Format】

```
ER ercd = net_dev_ini(UH dev_num);
```

【Parameter】

UH	dev_num	Device number

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

<0	Initialization failure

【Explanation】

Use dev_num to initialize a specific device. In fact, net_dev_ini uses dev_ini of driver device to initialize the device. If it completes normally, it can handle the packet through that network device.

net_dev_cls Release network device

【Format】

ER ercd = net_dev_cls(UH dev_num);

【Parameter】

UH	dev_num	Device number

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

<0	Failure

【Explanation】

Release specific device by using dev_num. In fact, net_dev_cls will release device by using dev_cls of device driver.

net_dev_ctl Network device control

【Format】

ER ercd = net_dev_ctl(UH dev_num, UH opt, VP val);

【Parameter】

UH	dev_num	Device number
UH	opt	Control code
VP	val	Value to set

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

<0	Failure

【Explanation】

Control the specific device by using dev_num. Because net_dev_ctl only calls dev_ctl of device driver, the actual actions depend much on integration of driver device.

net_dev_sts	Acquire the status of network device

【Format】

ER ercd = net_dev_sts(UH dev_num, UH opt, VP val);

【Parameter】

UH	dev_num	Device number
UH	opt	Status code
VP	val	Getting value

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

<0	Failure

【Explanation】

Acquire the status of specific device by using dev_num. Because net_dev_sts only calls dev_ref of device driver, the detailed action depends on integration of device driver.

5.4 Socket API (uNet3 compatible)

Application uses socket API to exchange TCP/UDP data with remote host.

When creating socket or connecting, it’s necessary to use device number to specify network device to connect. In case specify device number 0, it means that “Don’t specify device” , the interface selection action between socket and network device is different depending on transmission or reception. Besides, when creating socket, if it specify device number beside 0, it don’t need to specify device number when connecting.

For example, in the system is constructed by N network devices (N is more than 2) with the using socket APIs, show in the below it.

	Device number when creating (※1)	**Device number when connecting (※2)**	**Device in use**
Socket transmission action con_soc() of snd_soc() and TCP client (SYN transmission)	0	0	Device number 1(top)
	0	N	Device number N
	N	ANY	Device number N
Socket reception action con_soc() of rcv_soc() and TCP server (SYN reception)	0	0	Notified device (※3)
	0	N	Device number N
	N	ANY	Device number N

※1 Specify by host->num argument of con_soc() API.

※2 Specify by host->num argument of con_soc() API. In case of receiving through UDP socket, do not need to call con_soc() API.

※3 The socket which does not specify device number even when creating or connecting socket, if port number and protocol are matched, it can receive packet from any device. The socket in this case uses the device notified packet in subsequent operation.

cre_soc Socket generation

【Format】

```
ER ercd = cre_soc(UB proto, T_NODE *host);
```

【Parameter】

UH	proto	Protocol type
T_NODE	*host	Information of local host

【Return value】

ER	ercd	ID of generated socket (>0) or error code

【Error code】

E_ID	Unable to generate socket (exceed maximum number of socket)
E_PAR	‘host’ is wrong
E_NOSPT	‘proto’ is wrong

【T_NODE】

Specify local port number and device interface to use.

UH	port	Port number	Port number of local host. Specify the value from 1 to 65535 or PORT_ANY. In case that PORT_ANY is specified, it will determine port number by protocol stack.
UH	ver	IP version	Specify 0 (use IP_VER4)
UB	num	Device number	Specify device number of the device it want to use
UW	ipa	IP address	Specify 0 (use local IP address)

【proto】

Protocol type of socket to create

IP_PROTO_TCP	TCP socket
IP_PROTO_UDP	UDP socket

【Explanation】

This API creates the socket of specified protocol.

Example of creating TCP socket

```
T_NODE host;
host.num = 1;
host.port = 7;
host.ver = IP_VER4;
host.ipa = INADDR_ANY;
cre_soc(IP_PROTO_TCP, &host);
```

del_soc	Delete socket

【Format】

```
ER ercd = del_soc(SID sid);
```

【Parameter】

SID	sid	ID is used to identify socket

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Wrong ID number
E_NOEXS	Socket does not exist (Socket has not been created yet)
E_OBJ	Status of socket is wrong

【Explanation】

This API deletes the specified ID socket. When delete TCP socket, please call cls_soc() in advance and close the socket.

con_soc	Socket connection

【Format】

```
ER ercd = con_soc(SID sid, T_NODE *host, UB con_flg) ;
```

【Parameter】

SID	sid	ID is used to identify socket
T_NODE	*host	Information of remote host
UB	con_fig	Connection mode

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Wrong ID number
E_NOEXS	Socket does not exist (Socket has not been created yet)
E_PAR	Host or con_flg is wrong
E_OBJ	Socket status is wrong (for example, calling this API for the socket which has already connected)
E_TMOUT	Connection process is out of time
E_WBLK	Processed by non-blocking mode
E_CLS	Refuse the connection from remote host
E_RLWAI	Connection process is interrupted
E_QOVR	con_soc() is already executing

【T_NODE】

Specify remote host and device interface to use.

UH	port	Port number	Port number of remote host (from 1 to 65535)
UH	ver	IP version	Specify 0
UB	num	Device number	Device number of the device that wants to use
UW	ipa	IP address	IP address of remote host

【con_fig】

Specify the waiting for connection (server) or the active connection (client).

Usually specify 0 for UDP socket.

SOC_CLI	Connect to remote host (active connection)
SOC_SER	Wait for connection (passive connection)

【Explanation】

This API has different behavior depending on using protocol.

In the course of TCP, establish the connection to remote host, in case of UDP, associate the socket with the destination of data transmission.

An example for the connection of TCP server socket
T_NODE　remote = {0};　　　　　　/* clear by 0 */ con_soc(SID, &remote, SOC_SER);

An example for the connection of TCP client socket

T_NODE　remote;
remote.port = 100;　　　　　　　/* Port number of remote host */
remote.ver = IP_VER4;
remote.num = 1;　　　　　　　　/*Specify device number to use */
remote.ipa = ip_aton(“192.168.11.1”); /*IP address of remote host */
con_soc(SID, &remote, SOC_CLI);

cls_soc	Socket disconnetion

【Format】

ER ercd = cls_soc(SID sid, UB cls_flg);

【Parameter】

SID	sid	ID is used to identify socket
UB	cls_fig	Disconnetion mode

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Wrong ID number
E_NOEXS	Socket does not exist (Socket has not been created yet)
E_PAR	‘cls_flg is wrong
E_OBJ	Socket status is wrong (Such as when calling this API in the status of disconnection)
E_TMOUT	Disconnection process is out of time
E_WBLK	Processed by non-blocking mode
E_CLS	Forced termination of the connection from remote host
E_RLWAI	Disconnection process is interrupted
E_QOVR	cls_soc() is executing already

【cls_flg】

This parameter is only valid for TCP socket

SOC_TCP_CLS	Disconnect socket (End the connection)
SOC_TCP_SHT	Disable transmission process only. Reception is possible. (In case that want to stop the connection completely after using SOC_TCP_SHT to discontinue the transmission process only, it need to use SOC_TCP_CLS to disconnect completely.)

【Explanation】

This API has different behavior depending on using protocol.

In case of TCP, stop the connection to remote host, in case of UDP, clear the information of the destination or the source of data associated with socket. (After that, it cannot send UDP data).

cfg_soc Set parameter of socket

【Format】

ER ercd = cfg_soc(SID sid, UB code, VP val) ;

【Parameter】

SID	sid	ID is used to identify socket
UB	code	Parameter code
VP	val	Value to set

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Wrong ID number
E_NOEXS	Socket does not exist (Socket has not been created yet)
E_NOSPT	Wrong parameter code
E_PAR	Wrong parameter value
E_OBJ	Status of socket is wrong

【Explanation】

This API can set parameter as below. Please cast and then hand the setting value to VP type.

Example of setting

```
UB ttl = 32;
cfg_soc(SID, SOC_IP_TTL, (VP)ttl);
```

Parameter code	Data type	Meaning
SOC_TMO_CON	TMO	Calling timeout of con_soc
SOC_TMO_CLS	TMO	Calling timeout of cls_soc
SOC_TMO_SND	TMO	Calling timeout of snd_soc
SOC_TMO_RCV	TMO	Calling timeout of rcv_soc
SOC_IP_TTL	UB	Set TTL of IP header (Time to Live)
SOC_IP_TOS	UB	Set TOS of IP header (Type of Server)
SOC_CBK_HND	Pointer for function	Register callback function
SOC_CBK_FLG	UH	Set bit pattern of callback event flag (Value to set is as below)
SOC_PRT_LOCAL	UH	Change Local Port Number
SOC_UDP_RQSZ	UB	Change the receive queue size of the UDP socket (*)

(*) The receive queue size for UDP sockets is typically 1. If change this value, may lose packets that are already being received. If set the receive queue size to 0, packets will not be received.

Callback event flag bit	Meaning
EV_SOC_CON	Set con_soc() in non-blocking mode (only TCP socket)
EV_SOC_CLS	Set cls_soc() in non-blocking mode (only TCP socket)
EV_SOC_SND	Set snd_soc() in non-blocking mode
EV_SOC_RCV	Set rcv_soc() in non-blocking mode

Regarding callback event flag bit, it can set a multiple bit. In case of setting it multiple, set by OR. An example of setting is as below.

```
Ex:    ercd = cfg_soc(ID socket, SOC_CBK_FLG, (VP)(EV_SOC_CON|EV_SOC_SND|EV_SOC_RCV|EV_SOC_CLS));
```

Socket event set in non-blocking disables socket timeout of that event.

When enable callback event flag bit, it is necessary to register callback function in SOC_CBK_HND. Regarding callback function, please refer to the following.

soc_cbt Callback function

【Format】

UW soc_cbt(SID sid, UH event, ER ercd);

【Parameter】

SID	sid	ID is used to identify socket
UH	event	Callback event flag bit
ER	ercd	Error code

This callback function is called out from TCP/IP stack. However, in case of execution API socket of non-blocking mode, if API process is necessary to enter the waiting sate, it will return E_WBLK value without enter. This time, it will be notified from TCP/IP stack that the process of callback function has completed.

Call back event flag bit (event)	Error code (ercd)	Meaning
EV_SOC_CON	E_OK	con_soc() process completes normally
	< 0	con_soc() process completes with error. Regarding error content of this time, please refer to error code of con_soc().
EV_SOC_CLS	E_OK	cls_soc() process completes normally
	< 0	cls_soc() process complete with error. Regarding error content of this time, please refer to error code of cls_soc().
EV_SOC_SND	> 0	UDP socket : snd_soc()process completes normally TCP socket : In case of TCP transmission buffer is available, it will show the available size by ‘ercd’ value. Again, snd_soc() is called and then it can copy transmission data into TCP transmission buffer.
	<= 0	snd_soc() process completes normally. Regarding error content of this time, please refer to error code of snd_soc().
EV_SOC_RCV	> 0	UDP socket : There is receipt data in UDP socket. Shows receipt data size by ‘ercd’ value. Again, it can call rcv_soc() to receive data. TCP socket : There exists receipt data in TCP socket. Shows receipt data size by ‘ercd’ value. Again, it can call rcv_soc() to receive data.
	<= 0	rcv_soc() process completes normally. Regarding error content of this time, please refer to error code of rcv_soc().

※Prohibit to call all API functions of TCP/IP stack from callback function. (Please think callback unction the same as interrupting handler and use it).

ref_soc	Refer parameter of socket

【Format】

```
ER ercd = ref_soc(SID sid, UB code, VP val) ;
```

【Parameter】

SID	sid	ID is used to identify socket
UB	code	Parameter code
VP	val	Pointer for buffer of the value to get

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Wrong ID number
E_NOEXS	Socket does not exist (socket has not been created yet)
E_NOSPT	Wrong parameter code
E_PAR	Wrong parameter value (in case that val is NULL)
E_OBJ	Socket status is wrong (Cannot refer to socket)

【Explanation】

Refer the parameters as below. Please cast and then hand the setting value to VP type.

Example of getting remote host information

```
T_NODE remote;
ref_soc(SID, SOC_IP_REMOTE, (VP)&remote);
```

Parameter code	Data type	Meaning
SOC_TMO_CON	TMO	Calling timeout of con_soc
SOC_TMO_CLS	TMO	Calling timeout of cls_soc
SOC_TMO_SND	TMO	Calling timeout of snd_soc
SOC_TMO_RCV	TMO	Calling timeout of rcv_soc
SOC_IP_LOCAL	T_NODE	Getting Port number and IP address of local host
SOC_IP_REMOTE	T_NODE	Getting port number and IP address of remote host
SOC_IP_TTL	UB	Getting TTL (Time to Live)
SOC_IP_TOS	UB	Getting TOS (Type Of Service)
SOC_RCV_PKT_INF	T_RCV_PKT_INF	Get the latest packet information received by the UDP socket. In the case of TCP socket, get the connection destination information when the connection is established
SOC_PRT_LOCAL	UH	Reference Local Port Number

For the socket having both multicast address and unicast address, to know IP address of the last received packet, please refer to the below.

Example of getting received IP address

```
T_RCV_PKT_INF rcv_pkt_inf;
ref_soc(SID, SOC_RCV_PKT_INF, (VP)&rcv_pkt_inf);
if(rcv_pkt_inf.dst_ipa == MULTICASTADDRESS) {
    /* received by multicast address */
}
```

In the case of TCP socket, the connection destination information is set in the member variables of src_ipa and src_port, and the IP address and port number information of the local node is set in the member variables of dst_ipa and dst_port.

This reflects the information at the time when the TCP connection was established as the remote host information regardless of the server connection or client connection.

abt_soc Abort the socket process

【Format】

ER ercd = abt_soc(SID sid, UB code);

【Parameter】

SID	sid	ID is used to identify socket
UB	code	Control code

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_ID	Wrong ID number
E_NOEXS	Socket does not exist (socket has not been created yet)

【Explanation】

This API can cancel waiting status of con_soc, cls_soc, snd_soc, rcv_soc. Cancelled API returns E_RLWAI .

Control code	Meaning
SOC_ABT_CON	Discontinuation of con_soc() process
SOC_ABT_CLS	Discontinuation of cls_soc() process
SOC_ABT_SND	Discontinuation of snd_soc() process
SOC_ABT_RCV	Discontinuation of rcv_soc() process
SOC_ABT_ALL	Process discontinuation of all sockets

snd_soc Data transmission

【Format】

```
ER ercd = snd_soc(SID sid, VP data, UH len);
```

【Parameter】

SID	sid	ID is used to identify socket
VP	data	Pointer to the transmit data
UH	len	Size of the transmit data

【Return value】

ER	ercd	Actual transmitted data size (>0) or error code

【Error code】

E_ID	Wrong ID number
E_NOEXS	Socket does not exist (socket has not been created yet)
E_PAR	Wrong transmitted data or data size for transmission is not specified.
E_OBJ	Socket status is wrong
E_TMOUT	Transmission process is out of time
E_WBLK	Processed by non-blocking mode
E_CLS	TCP socket disconnected
E_RLWAI	Transmission process is interrupted
E_NOMEM	Memory is not enough
E_QOVR	snd_soc() is executing already
EV_ADDR	Destination default G/W unknown

【Explanation】

This API transmits data to remote host. When the process succeeds, it will return the actual transmitted data size. Besides that case, it will return error code.

In case of TCP socket, this API will copy data into protocol stack inside, and return that copied size. (Returned data size is less than len specified by argument) . Please refer to ”3.1.4 TCP module” for details.

In case of UDP socket, data will be transmitted to network and return that transmitted size. Please refer to “3.1.3 UDP module” for details.

rcv_soc Data reception

【Format】

```
ER ercd = rcv_soc(SID sid, VP data, UH len);
```

【Parameter】

SID	sid	ID is used to identify socket
VP	data	Pointer to receipt data
UH	len	Receipt data size

【Return value】

ER	ercd	Actual received data size (>0) or error code

【Error code】

E_ID	Wrong ID number
E_NOEXS	Socket does not exist (socket has not been created yet)
E_PAR	Wrong receipt data or receipt data size is not specified.
E_OBJ	Socket status is wrong
E_TMOUT	Receipt process timed out
E_WBLK	Processed by non-blocking mode
E_CLS	TCP socket disconnected
E_RLWAI	Reception process is interrupted
E_QOVR	rcv_soc() is executing already
0	Received up to the end of TCP socket data

【Explanation】

This API receive data which is sent from remote host.

In case of TCP, the maximum receivable size to receive is “Reception buffer size” specified by the configurator. Please refer to ”3.1.4 TCP module” for details.

In case of UDP, the maximum receivable size to receive is 1472 bytes (MTU default － IP header size － UDP header size). Please refer to “3.1.3 UDP module” for details.

soc_ext Stop process of socket all at once

【Format】

```
ER ercd = soc_ext(UH dev_num);
```

【Parameter】

UH	dev_num	Target device ID

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Invalid device ID

【Explanation】

Stops the operation of the socket during API execution or communication and returns the socket to the initial state.

Network device to be stopped is specified in the argument dev_num, but if DEV_ANY is specified, all sockets are targeted.

When the API is executed, the running socket API returns E_RLWAI.

For TCP sockets, the session will be reset during the session. In addition, not only TCP and UDP, but also send and receive packets held by the socket are released immediately.

After the API ends, the socket returns to the initial state (the state after socket generation), but various socket options set by the application such as callback, non-blocking, and timeout are retained.

5.5 Socket API (BSD compatible)

TCP/IP stack has the socket API for the BSD interface too. Thus, it is possible to easily divert the existing network applications using BSD sockets API.

5.5.1 Module overview

5.5.1.1 Position of the POSIX specification

TCP/IP stack is supported the socket API which is equivalent to the 4.4BSD-Lite. For supported API list, please refer the section **5.5.3 API list**.

5.5.1.2 The difference to uNet3 socket API

In addition to the POSIX-compliant sockets API, also provides the features that did not combine in μNet3 socket.

- Multiple call of socket API
- select() function
- Loop back address
- Multicast group with socket unit
- Listen queue of TCP socket
- Socket error

5.5.1.3 Compatibility of symbol name

The APIs, structures, and macros are added the unique prefix "unet3_" in order to avoid a symbol collision by the compiler environment.

By including sys/socket.h, the symbol name of the POSIX standard, which is used in application, be replaced with to these symbols with unique prefix. So application source codes based on BSD socket can be run as is.

The notation in this document has been using the POSIX standard symbol, because of considering the readability.

5.5.2 Module structures

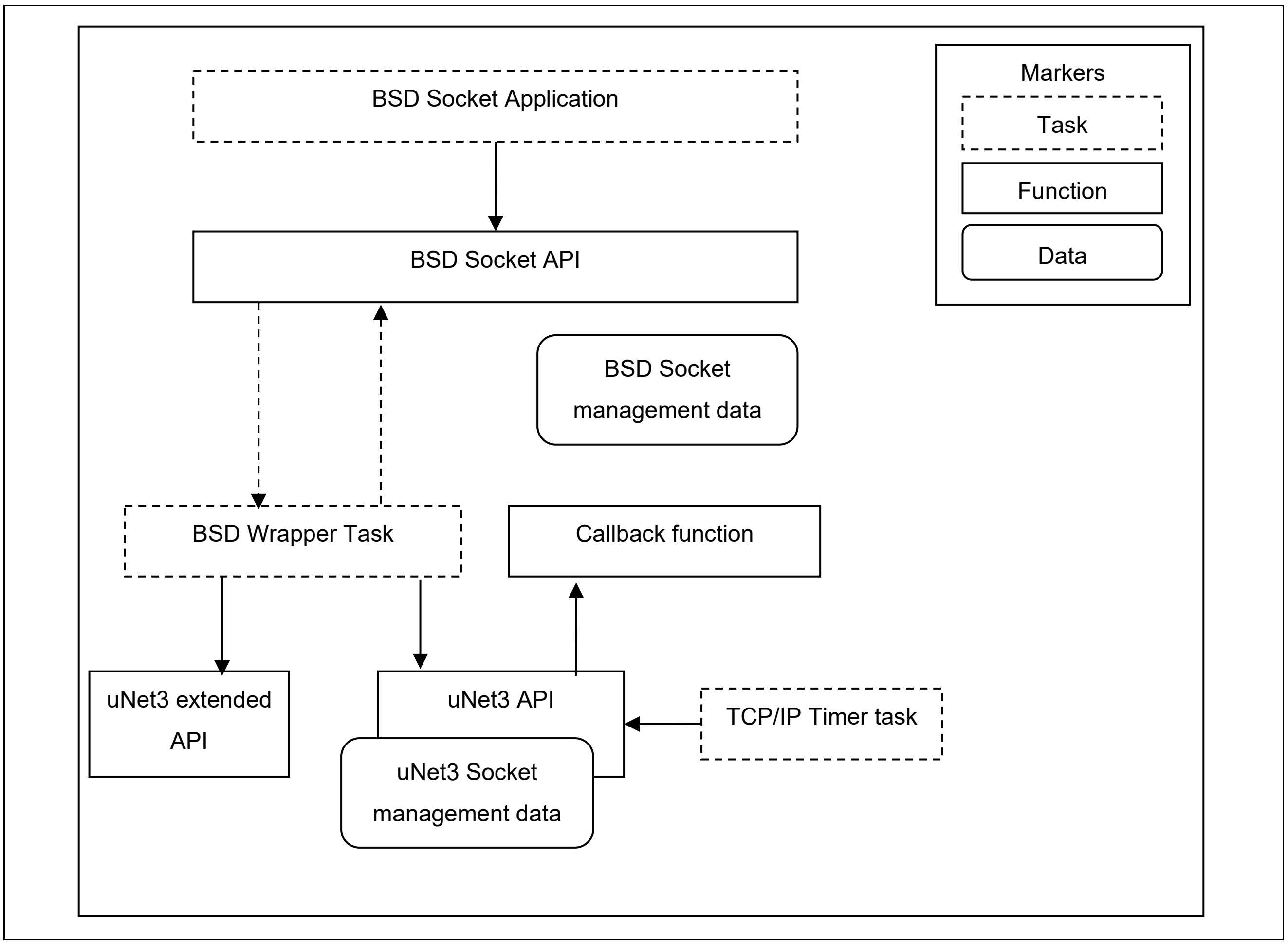


Fig. 5.1 Module structure for BSD socket

5.5.3 API list

Table 5.1 API list

API	Function	Include header
unet3_bsd_init	Initialize the TCP/IP and BSD stack	“sys/socket.h”
get_errno	Get the errno of each task	“sys/errno.h”
socket	Create an end point for communication	“sys/socket.h”
bind	Put a name for socket	“sys/socket.h”
listen	Wait the connection on the socket	“sys/socket.h”
accept	Accept connection to the socket	“sys/socket.h”
connect	Connect to the socket	“sys/socket.h”
send	Send message to the socket	“sys/socket.h”
sendto	Send message to the socket	“sys/socket.h”
recv	Receive message from the socket	“sys/socket.h”
recvfrom	Receive message from the socket	“sys/socket.h”
shutdown	Close the part of the full-duplex connection	“sys/socket.h”
close	Close the descriptor (socket)	“sys/ unistd.h”
select	Multiplexing of synchronous I / O	“sys/ select.h”
getsockname	Get the name of the socket	“sys/socket.h”
getpeername	Get the name of the peer socket	“sys/socket.h”
getsockopt	Get the socket options	“sys/socket.h”
setsockopt	Set the socket options	“sys/socket.h”
ioctl	Control the device (socket)	“sys/ioctl.h”
inet_addr	Internet address manipulation routines	“arpa/inet.h”
inet_aton	Internet address manipulation routines	“arpa/inet.h”
inet_ntoa	Internet address manipulation routines	“arpa/inet.h”
if_nametoindex	Mapping name and index of the network interface	“net/if.h”
if_indextoname	Mapping name and index of the network interface	“net/if.h”
rresvport	Get the socket which is bound to port	“sys/unistd.h”
getifaddrs	Get the address of the interface	“sys/types.h”
freeifaddrs	Release the interface information	“sys/types.h”
inet_pton	Convert IPv4/IPv6 addresses from text format to binary format	“arpa/inet.h”
inet_ntop	Convert IPv4/IPv6 addresses from binary format to text format	“arpa/inet.h”

5.5.4 Detail for each API

5.5.4.1 unet3_bsd_init

Please refer the section 5.5.7.7 Initialization

5.5.4.2 get_errno

Please refer the section 5.5.6.3 Error handling

5.5.4.3 socket (Create an end point for communication)

【Format】

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

【Parameter】

int	domain	Domain
int	type	Communication type
int	protocol	Protocol

【Return value】

int	Generated socket FD. On error, -1.

【errno】

ENOMEM	Over the generation possible number of sockets
	Message buffer is depleted
EINVAL	Invalid parameter
EINTR	Waiting state was forcibly released

【Explanation】

- The domain can be specified only AF_INET or AF_INET6.
- The communication type can be specified only SOCK_STREAM or SOCK_DGRAM.
- Protocol parameter is not used, so any value is possible.
- The number of sockets (sum of for TCP and UDP) that can be generated at the same time is the value defined by “#define CFG_NET_SOC_MAX”.
- The number of TCP sockets that can be generated at the same time is the value defined by “#define CFG_NET_ TCP _MAX”.
- The local port of socket cannot be set to 0. Therefore the socket immediately after generation is assigned the temporary local port number.

5.5.4.4 bind (Put a name for the socket)

【Format】

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

【Parameter】

int	sockfd	Socket FD
const struct sockaddr *	addr	Local address
unsigned int	addrlen	Local address length

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

EINVAL	Invalid parameter
ENOMEM	Message buffer is depleted
EBADF	Socket FD cannot be done "bind"
EPIPE	Socket FD is invalid
EAFNOSUPPORT	Unsupported address family
EADDRINUSE	Address is already in use
EADDRNOTAVAIL	Address is not available
EINTR	Waiting state was forcibly released

【Explanation】

- To start the receive operation, user needs to run the bind() function in advance with specifying the socket of the target. In here, the receive operation means a standby connection of TCP (listen()) and the reception of UDP packet (recv (), recvfrom ()).
- User can bind() to wellknown port (1-1023) also.

[AF_INET]

- Local address should be set as struct sockaddr_in type.
- IP address (IPv4) for the local address can be specified only the address which is set to device or INADDR_ANY (unspecified).
- If user sets the PORT_ANY (0) to the port number of the local address, a port number is assigned in the protocol stack.
- The local address length can be specified only sizeof(struct sockaddr_in) (=16).
- The member "sin_len" with struct sockaddr_in type is not used, so any value is possible.

[AF_INET6]

- Set the local address with struct sockaddr_in6 type.
- For the IP address (IPv6) of the local address, only the address set in the device or IN6ADDR_ANY_INIT (unspecified) can be specified.
- If specify a link-local address for the IP address (IPv6) of the local address, specify a network interface number other than 0 for the scope ID.
- If set PORT_ANY (0) for the port number of the local address, assign the port number in the protocol stack.

- Only sizeof (struct sockaddr_in6) (= 28) can be specified for the local address length.
- The value does not matter because the member "sin6_len" of type struct sockaddr_in6 is not used.
- Unlike POSIX, if the device specified by bind has multiple IPv6 addresses, the address specified by bind may not be selected depending on the address scope during socket communication.

.

5.5.4.5 listen (Wait the connection on the socket)

【Format】

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

【Parameter】

int	sockfd	Socket FD
int	backlog	Back log

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

EINVAL	Invalid parameters, connected TCP socket FD
ENOMEM	Message buffer is depleted
EBADF	Socket FD cannot be done "listen"
EPROTONOSUPPORT	Unsupported protocol (not TCP)
EINTR	Waiting state was forcibly released

【Explanation】

- TCP socket becomes to the state for waiting connection.
- The socket FD can be specified only the socket FD of TCP.
- The maximum number for back log is #define CFG_NET_TCP_MAX - 1.

5.5.4.6 accept (Accept connection to the socket)

【Format】

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

【Parameter】

int	sockfd	Socket FD
struct sockaddr *	addr	Remote address (output)
unsigned int *	addrlen	Remote address length (input/output)

【Return value】

int	The connected socket FD. On error, -1.

【errno】

EINVAL	Invalid parameter
ENOMEM	Message buffer is depleted
EBADF	The socket is not “listen”
EAGAIN	The socket is not connected. (During asynchronous execution)
ETIMEDOUT	Timeout (if the timeout is set)
EINTR	Waiting state was forcibly released

【Explanation】

- The socket FD can be specified only the socket FD of TCP that listen() function had been successful.
- The remote address is stored as struct sockaddr_in * type if the address family is AF_INET and struct sockaddr_in6 * type if the address family is AF_INET6.
- If the specified remote address length is too small, only the data for the specified address length is copied to addr, sockaddr_in type size (16 bytes) when the address family is AF_INET, and AF_INET6 when the address family is AF_INET. Stores the size of sockaddr_in6 type (28 bytes).
- If no socket is connected, accept () blocks processing until it is connected remotely.
- See 5.5.7.6 TCP socket errno use case for errno for TCP Sockets.

5.5.4.7 connect (Connect to the socket)

【Format】

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

【Parameter】

int	sockfd	Socket FD
const struct sockaddr *	addr	Remote address
unsigned int	addrlen	Remote address length

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

EINVAL	Invalid parameter
ENOMEM	Message buffer is depleted
EBADF	The socket cannot be “connect”
EHOSTUNREACH	Attempted to connect to an inaccessible node
ECONNREFUSED	The connection was refused
EAFNOSUPPORT	Unsupported address family
EISCONN	Already connected. Socket is under listen.
EALREADY	Already connected.
EAGAIN	The socket is under connection. (During asynchronous execution)
ETIMEDOUT	Timeout (if the timeout is set)
EINTR	Waiting state was forcibly released

【Explanation】

- The behavior or operation for connect() is different by the socket FD protocol or send/receive operation.
- In the case of the connection of TCP socket, SYN is sent to the remote address and try to do communication..
- In the case of the sending of UDP socket, the remote address is set as the destination address. However if the different destination address with sendto() is specified, the destination address for sendto() is used.
- If you a remote address family (sa_family) are specified to AF_UNSPEC, these settings will be cleared.
- Unlike the POSIX specification, there is no filtering function by remote address for the receiving operation of the UDP socket.
- Unlike the POSIX specification, in the connect() for TCP socket with the asynchronous setting, user cannot issue the connect() again after the connection is completed. For example after confirmed EAGAIN for connect() and be guaranteed the writable status by select(), sending/receiving data becomes possible, because TCP session has been established at that point in time.
- If connect () is issued for a TCP socket that is set asynchronously and then the connect () process ends, the behavior when the next connect () is issued is that the return value is 0 when the connection is successful, and errno is set last time. It will be the error number that was given. Conversely, if the connection fails, the return value = -1, and errno is the error number of the error cause. In addition, the connect () process that returns these results does not perform the connection process (handshake), so if you want to perform the connection process again, you need to issue connect () again.
- Use the socket option "SO_SNDTIMEO" to set a timeout for connect ().

5.5.4.8 send (Send message to the socket)

【Format】

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

【Parameter】

int	sockfd	Socket FD
const void *	buf	Address for sending data
unsigned int	len	Length for sending data
int	flags	Flag

【Return value】

int	Byte number which is sent. On error, -1.

【errno】

EINVAL	Invalid parameter (address is not set in buf)
ENOMEM	Message buffer is depleted
	Cannot get the network buffer corresponding to “len”, or the value of “len” is 0.
EBADF	The socket cannot be “send” (TCP not connected / CLOSED)
EPIPE	Socket FD is invalid
EDESTADDRREQ	Unsetting the address (UDP socket)
EACCESS	Broadcast refusal because broadcast transmission is not permitted
EAGAIN	Sending (during asynchronous execution)
ETIMEDOUT	Timeout (if the timeout is set)
EINTR	Waiting state was forcibly released

【Explanation】

- The length for sending data can be specified the value from 1 to 65535.
- The flag is not used, so any value is possible.
- The sending 0 byte UDP data cannot be sent, unlike the POSIX specification.
- If the USE_APLBUF bit is specified for the flag, transmission is possible regardless of the network buffer setting size.
- If specify the USE_APLBUF bit in the flag, the API will try to send until the len size transmission is complete or an error occurs, during which the application must ensure the consistency of the area indicated by buf.
- See 5.5.7.6 TCP socket errno use case for errno for TCP Sockets.

  (*) The lower layer queue refers to the IP layer address resolution processing or the link layer asynchronous transmission processing.

5.5.4.9 sendto (Send message to the 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);
```

【Parameter】

int	sockfd	Socket FD
const void *	buf	Address for sending data
unsigned int	len	Length for sending data
int	flags	Flag
const struct sockaddr *	dest_addr	The destination address
unsigned int	addrlen	Length of the destination address

【Return value】

int	Byte number which is sent. On error, -1.

【errno】

EINVAL	Invalid parameter (address is not set in buf)
ENOMEM	Message buffer is depleted
	Cannot get the network buffer corresponding to “len”
EBADF	The socket cannot be “sendto” (TCP not connected / CLOSED)
EPIPE	Socket FD is invalid
EDESTADDRREQ	Unsetting the address (UDP socket)
EACCESS	Broadcast refusal because broadcast transmission is not permitted
EAGAIN	The socket is under sending. (During asynchronous execution)
ETIMEDOUT	Timeout (if the timeout is set)
EINTR	Waiting state was forcibly released

【Explanation】

- The length for sending data can be specified the value from 1 to 65535.
- The flag is not used, so any value is possible.
- In the case of TCP socket, the destination address and length of it is not used
- The sending 0 byte UDP data cannot be sent, unlike the POSIX specification.
- If the USE_APLBUF bit is specified for the flag, transmission is possible regardless of the network buffer setting size.
- If specify the USE_APLBUF bit in the flag, the API will try to send until the len size transmission is complete or an error occurs, during which the application must ensure the consistency of the area indicated by buf.

  (*) The lower layer queue refers to the IP layer address resolution processing or the link layer asynchronous transmission processing.

5.5.4.10 recv (Receive message from the socket)

【Format】

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

【Parameter】

int	sockfd	Socket FD
void *	buf	Address for receiving buffer
unsigned int	len	Length for receiving buffer
int	flags	Flag

【Return value】

int	Byte number which is received (included 0). On error, -1.

【errno】

EINVAL	Invalid parameter (address is not set in buf)
ENOMEM	Message buffer is depleted
	Cannot get the network buffer corresponding to “len”
EBADF	The socket cannot be “recv” (TCP not connected / CLOSED)
EPIPE	Socket FD is invalid
EAGAIN	The packet had not received yet. (During asynchronous execution)
ETIMEDOUT	Timeout (if the timeout is set)
EINTR	Waiting state was forcibly released

【Explanation】

- If MSG_PEEK is specified for the flag, the packet is fetched without being deleted from the receive queue of the socket. So the next time you call an incoming call, you can get the same packet.
- If the USE_APLBUF bit is specified in the flag, reception is possible regardless of the network buffer setting size, but the application must guarantee the consistency of the area indicated by buf until the packet is received.
- The length for received buffer can be specified the value from 1 to 65535.
- If there are not any received packets, the recv() function blocks the processes until reception of packet.
- If not yet connected to the remote as TCP socket, recv () will result in an error.
- If in disconnected condition from remote with TCP socket, recv() returns 0.
- See 5.5.7.6 TCP socket errno use case for errno for TCP Sockets.

5.5.4.11 recvfrom (Receive message from the socket)

【Format】

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

【Parameter】

int	sockfd	Socket FD
void *	buf	Address for receiving buffer
unsigned int	len	Length for receiving buffer
int	flags	Flag
struct sockaddr *	src_addr	Source address
unsigned int *	addrlen	Length of source address

【Return value】

int	Byte number which is received (included 0). On error, -1.

【errno】

EINVAL	Invalid parameter (address is not set in buf)
ENOMEM	Message buffer is depleted
	Cannot get the network buffer corresponding to “len”
EBADF	The socket cannot be “recvfrom” (TCP not connected / CLOSED)
EPIPE	Socket FD is invalid
EAGAIN	The packet had not received yet. (During asynchronous execution)
ETIMEDOUT	Timeout (if the timeout is set)
EINTR	Waiting state was forcibly released

【Explanation】

- The length for received buffer can be specified the value from 1 to 65535.
- The flag is not used, so any value is possible.
- If there are not any received packets, the recvfrom() function blocks the processes until reception of packet.
- If not yet connected to the remote as TCP socket, recvfrom () will result in an error.
- If in disconnected condition from remote with TCP socket, recvfrom () returns 0.
- For the TCP socket, Source address and Length of source address is not used.
- The source address is stored as struct sockaddr_in * type if the address family is AF_INET and struct sockaddr_in6 * type if the address family is AF_INET6.
- If the specified source address length is too small, only the data for the specified address length is copied to src_addr, and if the address family is AF_INET, the size of sockaddr_in type (16 bytes), AF_INET6 If sockaddr_in6 type size (28 bytes) is stored.
- For TCP sockets, the source address and address length are not used.

5.5.4.12 shutdown (Close the part of the full-duplex connection)

【Format】

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

【Parameter】

int	sockfd	Socket FD
int	how	Direction for disconnection

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

EINVAL	Invalid parameter
ENOMEM	Message buffer is depleted
EBADF	The socket cannot be “shutdown”
EPIPE	Not be connected (TCP socket)
EINTR	Waiting state was forcibly released

【Explanation】

- The Direction for disconnection can be specified only SHUT_WR or SHUT_RDWR.

5.5.4.13 close (Close the descriptor (socket))

【Format】

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

【Parameter】

int	fd	Socket FD

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

EINVAL	Invalid parameter
ENOMEM	Message buffer is depleted
EBADF	The socket cannot be “close”
EINTR	Waiting state was forcibly released

【Explanation】

- If TCP socket had not disconnected yet, socket will close after disconnect TCP session.
- The closed socket FD cannot use until generate again.

5.5.4.14 select (Multiplexing of synchronous I/O)

【Format】

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

【Parameter】

int	nfds	The value that maximum value in the socket FD included in the readfds and writefds, plus 1
fd_set *	readfds	Socket FD set to monitor whether readable
fd_set *	writefds	Socket FD set to monitor whether writable
fd_set *	exceptfds	Socket FD set to monitor the exception (not supported)
struct timeval *	timeout	Monitor timeout

【Return value】

int	The number of the socket FD which is writable or readable. On timeout, 0. On error, -1.

【errno】

EINVAL	Invalid parameter
ENOMEM	Message buffer is depleted
EBADF	Socked FD that cannot be “select” is set.

【Explanation】

- The “exceptfds” is not used.
- Unlike the POSIX specification, in the execution of select() for the socket FD immediately after generation, it is writable for UDP socket (unreadable if not received the packet), or readable for TCP socket (unwritable).
- select () returns the number of socket FDs set by updating all three argument fdsets only when the status of the monitored socket (Read / Write only) is enabled. These arguments are not updated in the event of a timeout or error, so the application should first verify the return value.

5.5.4.15 getsockname (Get the name of the socket)

【Format】

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

【Parameter】

int	sockfd	Socket FD
struct sockaddr *	addr	Buffer to store the socket address
unsigned int *	addrlen	Size of buffer to store the socket address

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

EINVAL	Invalid parameter
ENOMEM	Message buffer is depleted
EBADF	The socket cannot be “getsockname”
EINTR	Waiting state was forcibly released

【Explanation】

- *addrlen must be set to the size of sockaddr_in (16 bytes or more) if the address family is AF_INET and the size of sockaddr_in6 (28 bytes or more) if the address family is AF_INET6.
- The socket address determined by the time that issued the following API.

  bind()

  connect()

  accept()

  send/sendto()

  recv/recvfrom()

  The socket address becomes the undefined value if these API failed.

5.5.4.16 getpeername (Get the name of the peer socket)

【Format】

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

【Parameter】

int	sockfd	Socket FD
struct sockaddr *	addr	Buffer to store the remote address
unsigned int *	addrlen	Size of buffer to store the remote address

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

EINVAL	Invalid parameter
ENOMEM	Message buffer is depleted
EBADF	The socket cannot be “getpeername”
ENOTCONN	The address is not set.
EINTR	Waiting state was forcibly released

【Explanation】

- *addrlen must be set to the size of sockaddr_in (16 bytes or more) if the address family is AF_INET and the size of sockaddr_in6 (28 bytes or more) if the address family is AF_INET6.
- In the case of TCP, user can get the remote address only for connected socket.
- In the case of UDP, user can get the remote address only for the socket which had set the address by “connect” or “sendto”, or the socket had already got a packet.

5.5.4.17 getsockopt (Get the socket options)

【Format】

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

【Parameter】

int	sockfd	Socket FD
int	level	Option level
int	optname	Option name
void *	optval	Buffer to store the value got
unsigned int *	optlen	Size of buffer to store the value got

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

EINVAL	Invalid parameter
ENOMEM	Message buffer is depleted
EBADF	The socket cannot be “getsockopt”
EPROTONOSUPPORT	Unsupported option
EINTR	Waiting state was forcibly released

【Explanation】

- SOL_SOCKET, IPPROTO_IP, IPPROTO_TCP, and IPPROTO_IPV6 can be specified for the option level.
- The option name that user can get for each option level, please refer to 5.5.5 Socket option.

5.5.4.18 setsockopt (Set the socket options)

【Format】

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

【Parameter】

int	sockfd	Socket FD
int	level	Option level
int	optname	Option name
const void *	optval	Buffer for the value to set
unsigned int	optlen	Size of buffer for the value to set

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

EINVAL	Invalid parameter
ENOMEM	Message buffer is depleted
EBADF	The socket cannot be “setsockopt”
EPIPE	Socket FD is invalid
EPROTONOSUPPORT	Unsupported option
EINTR	Waiting state was forcibly released

【Explanation】

- ・ SOL_SOCKET, IPPROTO_IP, IPPROTO_TCP, and IPPROTO_IPV6 can be specified for the option level.
- The option name that user can get for each option level, please refer to 5.5.5 Socket option

5.5.4.19 ioctl (Control the device (socket))

【Format】

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

【Parameter】

int	d	Socket FD
int	request	Request
...		Request parameter

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

EINVAL	Invalid parameter (address is not set in buf)
EBADF	Specified socket descriptor is invalid
ENOMEM	Message buffer is depleted
EFAULT	Cannot expand the request parameters
EINTR	Waiting state was forcibly released

- The following parameters can be set in the request.

Request Code		
FIONBIO	Non-blocking communication settings	1 (setting), 0 (release)
FIONREAD	Get the number of bytes of received packets held by the socket	(unsigned int *)&nread

- Regarding the non-blocking communication seeting, please refer to 5.5.6.1 Non-blocking setting
- For the value that can be acquired with the FIONREAD option, the received packet size (overall size) held in the receive window buffer is set in the case of a TCP socket. For UDP sockets, the size of the next received packet block (only at the beginning) is set.

5.5.4.20 inet_addr (Internet address manipulation routines)

【Format】

#include “arpa/inet.h”

unsigned int inet_addr(const char *cp);

【Parameter】

const char *	cp	The dot notation IP address

【Return value】

int	IP address binary value after conversion (network byte order)

【errno】

Not set

【Explanation】

- If the fail to convert happen, the return value is set to 0.

5.5.4.21 inet_aton (Internet address manipulation routines)

【Format】

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

【Parameter】

const char *	cp	The dot notation IP address
struct in_addr *	inp	Buffer to store IP address binary value after conversion (network byte order)

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

Not set

【Explanation】

- If the fail to convert happen, the return value is set to -1.

5.5.4.22 inet_ntoa (Internet address manipulation routines)

【Format】

```
#include “arpa/inet.h”
char *inet_ntoa(struct in_addr in);
```

【Parameter】

struct in_addr	in	IP address binary value (network byte order)

【Return value】

Int	The dot notation IP address after conversion

【errno】

Not set

【Explanation】

- The area that stores the converted dot notation IP address is stored in a statically allocated buffer and returned, so if you call this function again after this, the character string will be overwritten.

5.5.4.23 if_nametoindex (Mapping name and index of the network interface)

【Format】

```
#include “net/if.h”
unsigned int if_nametoindex(const char *ifname)
```

【Parameter】

const char *	ifname	Interface name

【Return value】

unsigned int	Interface index. On error, 0.

【errno】

ENXIO	Nonexistent interface name

【Explanation】

- The interface name is set by the device name (gNET_DEV[index-1].name [8]) to be used in the TCP/IP protocol stack.

5.5.4.24 if_indextoname (Mapping name and index of the network interface)

【Format】

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

【Parameter】

unsigned int	ifindex	Interface index
char *	ifname	Buffer for storing the interface name

【Return value】

int	Result of process. If success, ifname value, or on error, NULL.

【errno】

ENXIO	Nonexistent interface index

【Explanation】

- The interface name is set by the device name (gNET_DEV[index-1].name [8]) to be used in the TCP/IP protocol stack.

5.5.4.25 rresvport (Get the socket which is bound to port)

【Format】

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

【Parameter】

int *	port	Buffer to store port number

【Return value】

int	The socket FD which is bound to port. If socket is not existing, return -1.

【errno】

Not set

5.5.4.26 getifaddrs (Get the address of the interface)

【Format】

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

【Parameter】

struct ifaddrs**	ifap	The start address of the interface information list

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

ENOMEM	Error as getting the stored area of intereface information

【Explanation】

- This function gets the interface information for the part of the number of devices (CFG_DEV_MAX) that are set in the application as chain.
- If successfully completed, the following values are stored in “ifap”.

  (*ifap)->ifa_next : The pointer of the next structure of interface information is stored. If the last element, this field is NULL.

  (*ifap)->name : The pointer to the interface name is stored.

  (*ifap)->ifa_flags : The device number is stored.

  (*ifap)->ifa_addr : The IP address of the device is stored as the sockaddr type pointer.

  (*ifap)->ifa_netmask : The subnet mask of the device is stored as the sockaddr type pointer.

  (* ifap)-> ifa_ifu.ifu_broadaddr stores the device broadcast address as a sockaddr type pointer. If the IP address of (* ifap)-> ifa_addr is AF_INET6, (* ifap)-> ifa_ifu.ifu_broadaddr will be NULL.

  (*ifap)->ifa_ifu.ifu_dstaddr, (*ifap)->ifa_data : No use.
- After successful completion, user will need to release the interface information list in the freeifaddrs() function, because the interface information list is dynamically allocated.

5.5.4.27 freeifaddrs (Release the interface information)

【Format】

```
#include “sys/unistd.h”
void freeifaddrs(struct ifaddrs *ifa)
```

【Parameter】

struct ifaddrs*	ifap	The start address of the interface information list

【Return value】

void

【errno】

Not set

【Explanation】

- This function relase the interface information list which was got by getifaddrs() function.

5.5.4.28 inet_pton (Convert IPv4/IPv6 addresses from text format to binary format)

【Format】

```
#include “arpa/inet.h”
int inet_pton(int af, const char *src, void *dst);
```

【Parameter】

int	af	Address family
const char *	src	Network address string
void *	dst	Storage buffer for network address structure

【Return value】

int	Result of process. If success, 0 or on error, -1.

【errno】

EINVAL	Invalid parameter (address is not set in buf)
EAFNOSUPPORT	Unsupported address family

【Explanation】

- Only AF_INET and AF_INET6 can be specified as the address family. If you specify a value other than AF_INET or AF_INET6, -1 is returned as the return value.
- If the string specified in src is not the correct network address notation for the address family, dst will contain all 0s for each member of the network address structure and a return value of 1.

5.5.4.29 inet_ntop (Convert IPv4/IPv6 addresses from binary format to text format)

【Format】

```
#include “arpa/inet.h”
const char *inet_ntop(int af, const void *src, char *dst, unsigned int size);
```

【Parameter】

int	af	Address family
const void *	src	Network address structure
char *	dst	Conversion result string storage buffer
unsigned int	size	Storage buffer length of conversion result character string

【Return value】

const char *	Pointer to dst. NULL in case of error

【errno】

EINVAL	Invalid parameter (address is not set in buf)
EAFNOSUPPORT	Unsupported address family
ENOSPC	Insufficient storage buffer size

【Explanation】

- Only AF_INET and AF_INET6 can be specified as the address family.
- If the conversion fails, NULL is returned as the return value.

5.5.5 Socket option

The following figure shows the list of option which can be set or got by setsockopt() or getsockopt()API. If anything other than the specified options, The function setsockopt() and getsockopt() will return -1.

Table 5.2 Socket option

Option name	Type	Function
SOL_SOCKET level		
SO_ACCEPTCONN	int	Get the LISTEN state of TCP socket. For Get only.
SO_BROADCAST	int	Affect only to UDP broadcast transmission operation. For Get or Set.
SO_DOMAIN	int	Get the socket domain. For Get only.
SO_ERROR	int	Get the socket error. For Get only.
SO_KEEPALIVE (*1)	int	Enable the Keep-Alive function of TCP socket. For Set only.
SO_RCVBUF	int	Set the receive buffer. For TCP, it is a byte number of received window. For UDP, it is treated as a number of received packets (queue size). For Get or Set.
SO_RCVBUFFORCE	int	Same as for SO_RCVBUF.
SO_RCVTIMEO	timeval	Set the received timeout of socket. For Get or Set.
SO_SNDTIMEO	timeval	Set the sending timeout of socket. For Get or Set.
SO_TYPE	int	Get socket type. For Get only.
SO_REUSEADDR	int	Allow local port duplicate bind for UDP sockets. GET / SET.
IPPROTO_IP level		
IP_ADD_MEMBERSHIP	ip_mreqn	Join the multicast group. For UDP socket only. For Set only.
IP_DROP_MEMBERSHIP	ip_mreqn	Drop from the multicast group. For Set only.
IP_MTU	int	Get path MTU. For Get only.
IP_MULTICAST_TTL	int	TTL setting for multicast sending packet. For Get or Set.
IP_MULTICAST_IF	ip_mreqn	Device settings for multicast sending packet. For Get or Set.
IP_MULTICAST_LOOP	int	Loopback settings for multicast sending packet. For Get or Set.
IP_TOS	int	TOS setting for IP sending packet. For Get or Set.
IP_TTL	int	TTL setting for IP sending packet. For Get or Set.
IPPROTO_TCP level		
TCP_KEEPCNT (*1)	int	Set the number of times of TCP Keep-Alive probe. For Set only.
TCP_KEEPIDLE (*1)	int	Set the non-communication interval of TCP Keep-Alive started. For Set only.
TCP_KEEPINTVL (*1)	int	Set the sending interval for TCP Keep-Alive probe. For Set only.
TCP_MAXSEG	int	Set the MSS value of TCP socket. For Get or Set.
IPPROTO_IPV6 level		
IPV6_ADD_MEMBERSHIP	ipv6_mreq	Join a multicast group. Only valid for UDP sockets. SET only.
IPV6_DROP_MEMBERSHIP	ipv6_mreq	Multicast group withdrawal. SET only.

(* 1)TCP Keep Alive enable / disable (SO_KEEPALIVE) must be set before connecting TCP.
Also, TCP Keep Alive enable / disable and other Keep Alive settings are shared by all sockets.

5.5.6 Support function

5.5.6.1 Non-blocking setting

By using ioclt() function, user can set the call for socket API to non-blocking (or blocking). At the initial state, all of APIs are set as blocking. In the case of non-blocking setting, API may return -1, and set EAGAIN to error number. The following table shows API lists that non-blocking setting is available, the condition that error number becomes to EAGAIN, and the operation that application should behave.

The timeout setting by socket option doesn't affect to the API with non-blocking. And for the API of TCP/IP stack with BSD socket, the calling task may become the sleep state even under non-blocking setting, because of the spec through the inter-task communication.

Table 5.3 non-blocking API

API	Condition	Application behavior
connect	In the case of TCP socket, the return value is "-1", and error number is EAGAIN, always.	TCP socket continues to wait SYN/ACK from remote and resend SYN during the stipulated time, after returned -1. The TCP socket is monitored by writefds, because of becoming writable by select at the time of receiving the SYN / ACK. User does not need to run connect again after becoming writable.
accept	If there is no socket has been connected to the listen socket, the return value becomes "-1", and error number becomes EAGAIN.	The TCP socket is monitored by readfds, because of becoming readable by select at the time of receiving the SYN from remote. User should run accept again after becoming readable.
send sendto	In the case that it is TCP socket and the sending buffer is full, error number becomes EAGAIN. In the case that it is UDP socket and the socket is under sending, error number becomes EAGAIN.	EAGAIN means that the packet couldn't be sent by the condition of socket. (Then also packet is not sent.).
recv recvfrom	If packet has not received yet, error number becomes EAGAIN.	The socket is monitored by readfds, because of becoming readable by select at the time of receiving packet from remote. User should run recv again after becoming readable.

5.5.6.2 Loopback

If a local loopback address (127.0.0.1 ~ 127.255.255.254) is specified as the destination address, the packet is sent is notified to the network interface.

In the TCP/IP stack with BSD, loopback address does not have a specific device interface, it is treated as a send-only address. So, it is not possible to run the bind() function for loopback address.

5.5.6.3 Error handling

Symbol errno is the only global variables. This value is updated in accordance with the error that occurred during the API execution. If user wants to run the API from multiple tasks, it is recommended that user gets the last of errno that occurred within a task with using get_errno() function, in order to guarantee the integrity of errno.

【Format】

```
#include “sys/errno.h”
int get_errno(void)
```

【Parameter】

void

【Return value】

int	The last of errno that occurred within a task called from this API.

【errno】

Not set

【Explanation】

- errno of each task is stored in the global variable UW tsk_errno[] which is the application prepared. User needs to pre-set the maximum number of tasks in this array element.

Table 5.4 errorno list

errno	Value	Explanation
EINTR	4	API wait state was forcibly released
ENXIO	6	Interface is not existed.
EBADF	9	Socket FD is invalid.
ENOMEM	12	Out of memory
EACCESS	13	Deny access to requested processing
EFAULT	14	Parameter error
ENODEV	19	A fatal abnormality(or unknown) in the system
EINVAL	22	Parameter error
EPIPE	32	Socket object is invalid
EAGAIN	35	Run the blocking processing
EALREADY	37	Processing of already running
EDESTADDRREQ	39	Need a destination set
EPROTONOSUPPORT	43	Function is not supported
EAFNOSUPPORT	47	Address family is not supported
EADDRINUSE	48	Address is already in use
EADDRNOTAVAIL	49	Address is not available
EISCONN	56	Socket is already connected
ENOTCONN	57	Socket is not connected
ETIMEDOUT	60	Time out
ECONNREFUSED	61	Connection is denied
EHOSTUNREACH	65	Attempted to connect to an inaccessible node

5.5.7 Implementation of BSD application

5.5.7.1 Source code

Application which uses the BSD sockets needs to bind 4 files (unet3_iodev.c, unet3_option.c, unet3_socket.c, unet3_wrap.c) in the folder “Middleware/uNet3/bsd/” to the application project.

```
Middleware
  |
 +--uNet3
     +--bsd
        +--unet3_Iodev.c
        +--unet3_option.c
        +--unet3_socket.c
        +--unet3_wrap.c
```

In addition, user will need to link library for BSD called “libunet3bsd.a”.

```
Library
 |
 └────ARM or GCC or IAR
         +--libune3bsd.a       /* Library for BSD socket */
```

5.5.7.2 Include path

Application which uses the BSD sockets needs to add the directry “unet3_posix” and “inc” in the folder “Middleware/uNet3/bsd/” as a include path in the setting.

```
Middleware
  |
 +--uNet3
     +--bsd
        |
        +--unet3_posix   /* Include base folder for BSD socket */
        |   |   :
        |
        +--inc           /* Include base folder of uNet3 socket */
```

5.5.7.3 Configuration

In the TCP/IP stack with BSD, the maximum number of sockets and the number of the task of application is needed to macro defined in unet3_cfg.h, in advance.

The maximum number of sockets

```
#define     BSD_SOCKET_MAX
```

The maximum number of sockets, regardless of the protocol, shows the number of sockets that application generates at the same time (including also listen backlog). This macro definition is used for definition of the management table number of BSD socket and fd_set type, which will be described later. This value must be the same value as the µNet3 socket maximum number (CFG_NET_SOC_MAX).

Number of application tasks

```
#define     NUM_OF_TASK_ERRNO
```

The number of application tasks is the task number that can be generated in the kernel. This macro definition is used to the management table number of error number, which will be described later. This value, regardless of the use or non-use of BSD, should be set the number of generatable task.

5.5.7.4 Resource definition

Application which uses the BSD sockets needs to prepare a table for managing information.

BSD socket management table

```
T_UNET3_BSD_SOC    gNET_BSD_SOC[BSD_SOCKET_MAX];
```

BSD socket management table defines a global variable as T_UNET3_BSD_SOC-type array that the number of elements is BSD_SOCKET_MAX.

Error number management table

```
UW tsk_errno[NUM_OF_TASK_ERRNO];
```

Error number management table defines a global variable as T_UNET3_BSD_SOC-type array that the number of elements is NUM_OF_TASK_ERRNO.

5.5.7.5 Kernel objects

Kernel objects that TCP/IP stack with BSD uses, is as follows.

Resource name	Use	ID
Task	BSD Wrapper task	ID TSK_BSD_API
	Loopback device task	ID_LO_IF_TSK
Mailbox	Communication between the BSD Wrapper task	ID MBX_BSD_REQ
	Communication between the loopback device task	ID_LO_IF_MBX
Memory pool	Message buffer	ID MPF_BSD_MSG

5.5.7.6 TCP socket errno use case

If another API execution or communication event occurs during execution of send (), recv (), and accept () on the TCP socket and the original API that was waiting is terminated, the return value and errno at this time. Is shown in the table below.

Executing send (), recv (), and accept () means waiting for a free send buffer, waiting for a packet to be received, and waiting for a TCP passive connection, respectively.

Event	send		recv		accept	
	Return value	errno	Return value	errno	Return value	errno
Execute close ()	Continue waiting	—	-1	EBADF	-1	EINTR
Execute shutdown (WR)	Continue waiting	—	Continue waiting	—	Continue waiting	—
Execute shutdown (RDWR)	Continue waiting	—	0	—	Continue waiting	—
FIN reception	Continue waiting	—	0	—		
RST reception	-1	EBADF	-1	EBADF		
Socket options Timer expired	-1	ETIMEDOUT	-1	ETIMEDOUT	-1	ETIMEDOUT
TCP timer expired	-1	ETIMEDOUT	-1	ETIMEDOUT		

5.5.7.7 Initialization

Application needs to initialize the BSD module by calling the unet3_bsd_init() function, before using the BSD sockets API. In addition, before this BSD module initialization, the initialization of the uNet3 and device driver are needed to be completed successfully.

【Format】

```
#include "sys/socket.h"
ER unet3_bsd_init(void)
```

【Parameter】

void

【Return value】

ER	Result of process. If success, E_OK. or if error, error code.

【errno】

E_SYS	Failed to initialize the process of kernel objects

Example of use

```
ER net_sample(void)
{
    /* Initialize TCP/IP Stack */
    ER ercd;

    ercd = net_ini();
    if (ercd != E_OK) {
        return ercd;
    }

    /* Initialize Ethernet Driver */
    ercd = net_dev_ini(1);
    if (ercd != E_OK) {
        return ercd;
    }

    /* BSD wrapper */
    ercd = unet3_bsd_init();

    return ercd;
}
```

5.6 Other API

htons	Convert 16 bit value to network byte order

【Format】

UH htons(UH val);

【Parameter】

UH	val	16 bit value host byte order

【Return value】

UH		16 bit value to network byte order

htonl	Convert 32 bit value to network byte order

【Format】

UW htonl(UW val);

【Parameter】

UW	val	32 bit value host byte order

【Return value】

UW		32 bit value network byte order

ntohs	Convert 16 bit value to host byte order

【Format】

UH ntohs(UH val);

【Parameter】

UH	val	16-bit value network byte order

【Return value】

UH		16-bit value host byte order

ntohl	Convert 32 bit value to host byte order

【Format】

UW ntohl(UW val);

【Parameter】

UW	val	32 bit value network byte order

【Return value】

UW		32 bit value host byte order

ip_aton	Convert an IPv4 address string in dot-notation to 32 bit value

【Format】

UW ip_aton(const char *str);

【Parameter】

char	str	Pointer to IPv4 address string in dot-notation

【Return value】

UW	>0	Successful completion (32 bit value after converting)

【Error code】

0	Wrong IP address is specified

ip_ntoa	Convert 32-bit value IPv4 address to IPv4 address string in dot-notation

【Format】

void ip_ntoa(const char *str, UW ipaddr);

【Parameter】

char	str	Pointer that accepted IP address string after converting
UW	ipaddr	32-bit value IP address

【Return value】

None

【Explanation】

If the process completes successfully, the character string will be set in str, but str will be NULL if the error occurs.

ip_byte2n	Convert IPv4 address array to 32 bit value

【Format】

UW ip_byte2n(char *ip_array);

【Parameter】

char	ip_array	Pointer to byte value array of IP address

【Return value】

UW	>0	Successful completion (32-bit value after converting)

【Error code】

0	Wrong IP address is specified

ip_n2byte	Convert 32 bit value IPv4 address to array

【Format】

void ip_n2byte(char *ip_arry, UW ip);

【Parameter】

char	ip_array	Pointer to byte value array of IP address
UW	ip	32-bit value IP address

【Return value】

None

【Explanation】

After completing successfullyl, value is set in asip_array. In case of error, ip_array turns into NULL.

arp_set Setting static ARP entries

【Format】

```
ER ercd = arp_set(UH dev_num, UW ip, UB *mac);
```

【Parameter】

UH	dev_num	Device number
UW	ip	IPv4 address
UB	*mac	MAC address

【Return value】

ER	ercd	Normal termination (E_OK) or error code

【Error code】

E_ID	Illegal device number
E_PAR	Illegal IPv4 address and MAC address
E_NOMEM	ARP cache is full

【Explanation】

Register a static entry in the ARP cache. For dev_num, specify the device number that connects to the network where the address exists.

Set the IP address and MAC address of the corresponding host for ip and mac, respectively. The MAC address is set in a UB type 6-byte array with big endianness. E_NOMEM is returned if the ARP cache is already full due to dynamic or other static entries. In that case, the application should set an entry or increase the size of the ARP cache before communication.

arp_ref ARP cache reference

【Format】

```
ER ercd = arp_ref(UH dev_num, UW ip, UB *mac);
```

【Parameter】

UH	dev_num	Device number
UW	ip	Search target IPv4 address
UB	*mac	Search result MAC address

【Return value】

ER	ercd	Normal termination (E_OK) or error code

【Error code】

E_ID	Illegal device number
E_NOEXS	There is no corresponding address

【Explanation】

Refers to the MAC address in the ARP cache. For dev_num, specify the device number that connects to the network where the address exists.

For ip, specify the IP address of the referenced host. If the corresponding IP address is found, copy the 6-byte MAC address to the argument mac.

arp_req Send ARP request

【Format】

```
ER ercd = arp_req(UH dev_num, UW ip);
```

【Parameter】

UH	dev_num	Device number
UW	ip	Search target IPv4 address

【Return value】

ER	ercd	Normal termination (E_OK) or error code

【Error code】

E_ID	Illegal device number
E_OBJ	The device itself does not have an address (not started)
E_NOMEM	Network buffer cannot be secured
E_PAR	Address cannot be resolved on the corresponding device (loopback, PPP, etc.)

【Explanation】

Application sends an ARP request at any time. This process does not wait for an ARP response.

When a normal ARP response is obtained, the response content is saved in the ARP cache. Applications can reference this using arp_ref ().

arp_clr Clear ARP cache

【Format】

```
ER ercd = arp_clr(void);
```

【Parameter】

None

【Return value】

ER	ercd	Normal termination (E_OK)

【Error code】

None

【Explanation】

Deletes all ARP cache entries. The application can clear the ARP cache when the network changes physically (after link down / link up is detected) or when the IP address changes.

arp_del Delete ARP entry

【Format】

```
ER ercd = arp_del(UH dev_num, UW ip);
```

【Parameter】

UH	dev_num	Device number
UW	ip	Delete target IPv4 address

【Return value】

ER	ercd	Normal termination (E_OK) or error code

【Error code】

E_ID	Illegal device number

【Explanation】

Deletes the statically configured ARP entry from the cache. The entry to be deleted is the specified IP address and statically registered by the application. Dynamically retrieved entries are not deleted.

6. Network application

6.1 DHCP client

DHCP client obtains IP address information which is used in network from DHCP server. Acquired IP address is assigned to the host.

SE THIS FUNCTION OF RENEW, RELEASE, DECLINE, FEATURES INFORM. FOR DHCP EXTENDED VERSION, SEE THE EXTENDED VERSION "6.5 DHCP client extended".

(1) Host address information

```
typedef struct t_host_addr {
        UW      ipaddr;             /* IP address */
        UW      subnet;             /* Subnet Mask */
        UW      gateway;            /* Gateway*/
        UW      dhcp;               /* DHCP Server address */
        UW      dns[2];             /* DNS Address */
        UW      lease;              /* Lease period of the DHCP address*/
        UW      t1;                 /* Renewal period of DHCP address*/
        UW      t2;                 /* Rebind period of DHCP address */
        UB      mac[6];             /* MAC address */
        UH      dev_num;            /* Device address */
        UB      state;              /* DHCP Cliente status*/
        SID     socid;              /* ID of UDP socket*/
} T_HOST_ADDR ;
```

This structure is used as an argument of DHCP client API. **Device number** and **ID of UDP socket** have to be set by user application. The remaining parameters are set by the response from a DHCP server.

- ID of UDP socket

In DHCP client, use UDP socket. UDP socket must be created by the following parameter. (It is created in DHCP client applications)

Protocol	ID	Port	Transmission timeout	Reception timeout
UDP	ID_SOC_DHCP	68	3 seconds	3 seconds

- Device number

In device number, specify network device used by DHCP client. If specify '0 ', default network device is used

6.1.1 DHCP client API

dhcp_client	Starting DHCP Client

【Format】

```
ER ercd = dhcp_client(T_HOST_ADDR *addr);
```

【Parameter】

T_HOST_ADDR	*addr	host address information

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	*addr is NULL or socid is specified
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	Socket status is wrong (socket has not been created yet)
E_TMOUT	Response from DHCP server delays. Or DHCP server does not exist

【Explanation】

This API acquires an IP address, a subnet mask, a gateway address from a DHCP server and assigns them to host. Maybe E_TMOUT error occurs due to the construction of using network. At that time, we recommend that you try to retry until it succeeds.

This API will also initiate a new DHCP session. When you call the API, you will start to send operation DISCOVER always expect receive OFFER, send REQUEST, the reception of the ACK that is.

The expiration of an IP address which is acquired from a DHCP server is specified 'lease (lease period).
As follows: DHCP client to do lease before the lease period expires.

DHCP client example (Exclusive task)

```
void dhcp_tsk(VP_INT exinf)
{
    ER ercd;
    T_HOST_ADDR dhcp_addr = {0};
    UB status = DHCP_STS_INIT;

    dhcp_addr.socid = ID_SOC_DHCP;
    dhcp_addr.dev_num = ID_DEVNUM_ETHER;

    for (;;) {
        ercd = dhcp_client(&dhcp_addr);
        if (ercd == E_OK) {
            /* BOUND period */
            dly_tsk(dhcp_addr.t1*1000);
            /* RENEWING period */
            status = DHCP_STS_RENEWING;
            continue;
        }
        if (status == DHCP_STS_RENEWING) {
            /* REBINDING period */
            dly_tsk((dhcp_addr.t2-dhcp_addr.t1)*1000);
            status = DHCP_STS_INIT;
            continue;
        }
        /* INIT period */
        dly_tsk(1000);
    }
}
```

If it is need to update the lease on the DHCPREQUEST message, please use the extended version of DHCP CLient.

6.2 FTP Server

FTP server enables to download and upload files to the remote host.

(1) FTP Server Control Information

```
typedef struct t_ftp_server {
        ·      ·
        UW     sec;                  /* Security policy */
        ·      ·
        ·      ·
        UH     dev_num;              /* Device Number*/
        SID    ctl_sid;
        SID    dat_sid;              /* ID Socket used for data*/
        ·      ·
        ·      ·
        ER (*auth_cbk)(UH, const char*, const char*);    /* Authentication callback */
        VB*    syst_name;            /* SYST command response string*/
} T_FTP_SERVER ;
```

Set necessary information in this structure and then transfer as argument of FTP Server API.

- Security policy

Set security settings in the security policy. If "0" is set, the security function will not be used.
If "ENA_DENY_PORTCOMMAND" is set, the PORT command rejection function is applied.
If "ENA_NOTCON_WELL_KNOWNPORT" is set, connection refusal from the Well-known port will be applied.
If apply the PORT command deny function, you will not necessarily be able to connect on the Well-known port. Therefore, when "ENA_DENY_PORT COMMAND" is set, the function of "ENA_NOTCON_WELL_KNOWNPORT" is automatically enabled. If you want to respond to the SYST command, set "ENA_ALLOW_SYSTCOMMAND".

ENA_DENY_PORTCOMMAND
Deny PORT command function

ENA_NOTCON_WELL_KNOWNPORT
Deny Well-known port connection

ENA_ALLOW_SYSTCOMMAND
Enable SYST command. Returns the value of T_FTP_SERVER :: syst_name to the query source. If the value is not set, "UNIX Type: L8" is returned to the inquiry source.

- Device Number

In device number, specify network device used in FTP server. In case of specifying “0”, default network device will be used (Please set 0 normally)

- TCP socket

FTP server requires two TCP sockets for commands and data. TCP socket should be created in the following parameters. (It will be created in the FTP server application)

Socket used for command:

ID	Protocol	Port	Timeout				Buffer size	
			send	receive	connect	interrupt	send	receive
ID_SOC_FTP_CTL	TCP	21	5s	15s	-1s	5s	1024	1024

Socket used for data:

ID	Protocol	Port	Timeout				Buffer size	
			send	receive	connect	interrupt	send	receive
ID_SOC_FTP_DATA	TCP	20	5s	15s	5s	5s	1024	1024

- Authentication callback

When performing arbitrary user authentication processing, specify the callback function in this variable. Called when authenticating after entering the user name and password. Normally, 0 is fine.

- Configuration definition (ftp_server_cfg.h)

Example setting

```
#include "ffsys.h"      /* File system */

/* Configuration */
#define CFG_FTPS_DRV_NAME          'C'                  /* Drive name */
#define CFG_FTPS_PATH_MAX          PATH_MAX             /* Maximum length of file path */
#define CFG_FTPS_CMD_TMO           5000                 /* 5 sec */
#define CFG_FTPS_DAT_TMO           5000                 /* 5 sec */
#define CFG_FTPS_IDLE_TMO          (5 * 60 * 1000)      /* 5 minute */
#define CFG_FTPS_SES_NUM           1                    /* Number of sessionss */
```

CFG_FTPS_DRV_NAME

Specify the drive name (such as 'A' or 'C') used by the file system. The FTP server uses the drive name of this macro to open the file.

CFG_FTPS_PATH_MAX

Specifies the maximum file path length for the file system. The path length here represents the length of the root directory name + file name character string.

CFG_FTPS_CMD_TMO

Specifies the send and disconnect timeout (msec) value for the control socket.

CFG_FTPS_DAT_TMO

Specifies the data socket connection, send, receive, and disconnect timeout (msec) values.

CFG_FTPS_IDLE_TMO

Specify the value of the receive timeout (msec) of the control socket.

CFG_FTPS_SES_NUM

Specify the number of tasks (sessions) on the FTP server.

- Account settings (ftp_server_cfg.c)

Example setting

```
#include "kernel.h"
#include "net_hdr.h"
#include "ftp_server.h"

/* Login user table (Max. 256 users) (DEV_ANY: All device is allowed) */
const T_FTP_USR_TBL ftp_usr_tbl[] = {
    {DEV_ANY, "", ""},                  /* Anyone can login (No user name,password) */
    {DEV_ANY, "User", "Password"},

    {0x00, 0x00, 0x00}                  /* Terminate mark (Do not change) */
};
```

Declare the array variables of the structure that sets the account in the file ftp_server_cfg.c. The structure for setting up an account is as follows.

```
typedef struct t_ftp_usr_tbl {
        UH      dev_num;            Device number
        VB*     usr;                User name
        VB*     pwd;                Password
} T_FTP_USR_TBL ;
```

The dev_num of the structure specifies the number of the network device to use when authenticating the account.
If 0 (DEV_ANY) is specified, all network devices will be authenticated. Generally, substitute 0 for dev_num. usr specifies the user name as a character string. pwd specifies the password as a string.
Declare the array variable of this structure with the variable name of ftp_usr_tbl. The last element of the array is the data for termination. For the data for termination, assign 0 to all variables. Up to 256 user names and passwords can be registered. In the above setting example, "{DEV_ANY," "," "}" is a setting that allows you to log in to the FTP server without entering the user name and password. Therefore, please do not register for any purpose other than testing.

6.2.1 FTP Server API

ftp_server	Start up FTP Server

【Format】

```
ER ercd = ftp_server(T_FTP_SERVER *ftp);
```

【Parameter】

T_FTP_SERVER	*ftp	FTP server control information

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Wrong parameter is specified (*fpt is NULL. ctl_sid or dat_sid is not specified)
E_RLWAI	Request to stop the FTP server has been executed.

【Explanation】

This API initializes FTP server, accepts and processes requests from FTP clients. Because this API becomes blocking calling, please use specific task to call it.

ftp_server_stop	Stop FTP server

【Format】

```
ER ercd = ftp_server_stop( UW retry );
```

【Parameter】

UW	retry	Number of retries for server task stop processing

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_TMOUT	Retry over for server task stop processing has occurred.

【Explanation】

This API will stop the FTP server that is running. When this API is executed, it will stop all tasks on the FTP server.

The task information of the FTP server is automatically stored in the internal memory when the FTP server startup API (ftp_server) is executed.

In addition, this API returns E_TMOUT as a return value when a retry over of the specified number of retries (retry) occurs at the time of execution.

【Recommendation】

It is recommended that the value of the number of retries (retry) specified by this API is "5" or more. For example, if you specify "0" for the number of retries and execute it, the probability that this API will fail is higher.

6.2.2 Restriction terms

- Operation on IPv6 and SSL is not supported. (IPv4 only)
- File system is required to handle files.

6.3 HTTP server

HTTP server transmits content to HTTP client (internet browser) statically and dynamically.

(1) HTTP content information

```
typedef struct t_http_file {
        const char      *path;                          /* URL */
        const char      *ctype;                         /* Content type */
        const char      *file;                          /* Content */
        Int             len;                            /* Content size*/
                                                        /* HTTP callback function
        void(*cbk)(T_HTTP_SERVER *http);                or
                                                        CGI handler */
        UB              ext;                            / * Extended operation flag * /
} T_HTTP_FILE ;
```

Register content to be used in HTTP server in this structure.

- URL

Show URL of content. For example, if there is request from client to that URL, corresponding content will be sent to client.

Impossible to specify NULL in URL. Besides, URL starts by `/ as usual.

- Content type

Specify Content-Type of text/html etc. In case of dynamic content, specify NULL.

- Content

Specify actual content. In case of dynamic content, specify NULL.

- Content size

Specify size of content. In case of dynamic content, specify 0.

- Callback function or CGI handler

When it's dynamic content, specify pointer of the function called from HTTP server. In case of static content, specify NULL.

- Extended operation flag

Specifies extended behavior for the content. Specify 0 if you do not want to use extended behavior (use HTTP authentication, add arbitrary HTTP headers, etc.).

```
#define HTTPD_EXT_AUTH          0x01          / * Use HTTP authentication * /
#define HTTPD_EXT_UHDR          0x02          / * Custom Header-Use * /
```

Defined value in T_HTTP_FILE :: ext

(2) HTTP server control information

```
typedef struct t_http_server {
        UW                  sbufsz;           /* transmission buffer size */
        UW                  rbufsz;           /* reception buffer size*/
        UW                  txlen;            /*internal data */
        UW                  rxlen;            /* internal data*/
        UW                  rdlen;            /* internal data*/
        UW                  len;              /* internal data*/
        UB                  *rbuf;            /* transmission buffer*/
        UB                  *sbuf;            /* reception buffer*/
        UB                  *req;             /* internal data*/
        UH                  Port;             /* Listening port number*/
        SID                 SocketID;         /* Socket ID */
        T_HTTP_HEADER       hdr;              /* HTTP client request */
        UB                  NetChannel;       /* Device number*/
        UB                  ver;              / * IP version * /
        UB                  server_tsk_stat   / * HTTP server task startup status * /
        ID                  server_tsk_id     / * HTTP server task ID * /
        struct t_http_server *next            / * HTTP server object * /
        UH                  kpa_max           / * HTTP KeepAlive max value (for control) * /
} T_HTTP_SERVER;
```

This structure is used as argument of HTTP server API. ID socket needs to be set by user application.

- Device number

For device number, specify network device to use in HTTP server. In case of specifying “0”, default network device is used. (Please set “0” normally).

- Socket ID

In HTTP server, use TCP socket. TCP socket needs to be created by the following parameter. (It will be created in HTTP server application).

ID	protocol	port	timeout				Buffer size	
			send	receive	connect	interrupt	send	receive
ID_SOC_HTTP	TCP	80	25s	25s	25s	25s	1024	1024

- Transmit buffer Receive buffer

In the HTTP server uses the network protocol stack buffers for each send and receive packets.

And receive buffers (transmit), if you (for example, you want to send content to a larger network buffers, for example) for reasons such as content size, you want to use your own buffer application in which the value of the buffer own buffer size (send) and receive set. You will not get the network buffer on the HTTP server in that case.

Own area set cannot be shared with other processes HTTP server also.

(3) HTTP Header information

```
typedef struct t_http_header {
        char            *method;        /* Method    */
        char            *url;           /* URL */
        char            *url_q;         /* URL query */
        char            *ver;           /* Version */
        char            *host;          /* Host */
        char            *ctype;         /* Content type */
        char            *Content;       /* Content    */
        char            ContentLen;     /* Content length */
        char            kpa;            /* Flag for HTTP Keep Alive (for control) */
        char            *auth;          /* Authorization header    */
        char            *cookie;        /* Cookie header */
}T_HTTP_HEADER;
```

This structure represents the buffer of T_HTTP_SERVER :: rbuf as each element of the HTTP request message. See this structure only from HTTP callback functions.

・ Method [method]
Represents a method. Enter either "GET", "HEAD", or "POST".

• Pathname [URL]
Represents the pathname of the URL.

• URL query
Represents a URL query parameter.

• Version [version]
Represents the HTTP version. Either "HTTP / 1.1" or "HTTP / 1.0" will be entered.

- Configuration definition (http_server_cfg.h)

```
Example setting
#define ENA_KEEP_ALIVE              /* Enable HTTP Keep-Alive */
#define HTTP_KPA_MAX     100      /* Keep-Alive max value */
#define ENA_USER_EXT               /* Enable User Extension */
```

• ENA_KEEP_ALIVE
Enables the HTTP Keep-Alive feature by definition. Disable without definition.

• HTTP_KPA_MAX
Specifies the initial value of the max item in the HTTP Keep-Alive header.

• ENA_USER_EXT
Enables the user extension operation function of the HTTP server application with definition. Disable without definition.

6.3.1 HTTP server API

http_server	Start up HTTP server

【Format】

```
ER ercd = http_server(T_HTTP_SERVER *http);
```

【Parameter】

T_HTTP_SERVER	*http	HTTP server control information

【Return value】

ER	ercd	Error code

【Error code】

E_RLWAI	http_server_stop () was called and stopped.
E_PAR	Wrong parameter is specified (*http is NULL. SocketID is not specified.)
E_NOMEM	Insufficient memory (exhausted network buffer)
Others	Socket con_soc error code

【Explanation】

This API initializes HTTP session, then accepts and processes request from HTTP client. In case the URL which is requested from client exists in content table (T_HTTP_FILE), send that content to client, if not, send HTTP error message" 404 File not found" . In case content is dynamic (cbk is not NULL), call that callback function.

Because this API becomes blocking call, please use a specific task to call it.

If NULL, the receive buffer of the control information of the argument is HTTP server uses the network buffer.

If NULL, the transmit buffer of the control information of the argument is HTTP server uses the network buffer.

http_server_stop	Stop HTTP server

【Format】

ER ercd = http_server_stop(UW retry)

【Parameter】

UW	retry	Number of retries for server task stop processing

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_TMOUT	Retry over for server task stop processing has occurred.

【Explanation】

This API stops the HTTP server that is running. When this API is executed, all tasks of the HTTP server will be stopped.

HTTP server task information is automatically stored in internal memory when the HTTP server startup API (http_server) is executed.

In addition, this API returns E_TMOUT as a return value when a retry over of the specified number of retries (retry) occurs at the time of execution.

【Recommendation】

It is recommended that the value of the number of retries (retry) specified by this API is "5" or more. For example, if you specify "0" for the number of retries and execute it, the probability that this API will fail is higher.

CgiGetParam CGI argument analysis

【Format】

```
void CgiGetParam(char *msg, int clen, char *cgi_var[], char *cgi_val[], int *cgi_cnt);
```

【Parameter】

char	*msg	CGI argument
int	clen	CGI argument size
char	*cgi_var[]	Analysed CGI argument
char	*cgi_val[]	Value of analysed CGI argument
int	*cgi_cnt	Number of articles of the analysed CGI argument

【Return value】

None

【Error code】

None

【Explanation】

This API analyses query string to be constructed in “field value” group. For example, analysis result in case of query string is given as “name1=value1&name2=value2” as below.

```
cgi_cnt = 2;

cgi_var[0] = “name1”;
cgi_var[1] = “name2”;

cgi_val[0] = “value1”;
cgi_val[1] = “value2”;
```

[Supplement]

This API has been superseded by the CgiGetParamN function. This API causes a buffer overrun when the number of elements obtained by parsing the query string of the argument msg exceeds the number of arrays of cgi_var and cgi_val, so use the CgiGetParamN function unless there is a special reason. Please.

CgiGetParamN function will be deprecated in the future.

CgiGetParamN CGI argument analysis

【Format】

```
void CgiGetParamN(char *msg, int clen, char *cgi_var[], char *cgi_val[], int *cgi_cnt);
```

【Parameter】

char	*msg	CGI argument
int	clen	CGI argument size
char	*cgi_var[]	Analysed CGI argument
char	*cgi_val[]	Value of analysed CGI argument
int	*cgi_cnt	IN : Number of elements in the cgi_var array OUT : Number of articles of the analysed CGI argument

【Return value】

None

【Error code】

None

【Explanation】

This API analyses query string to be constructed in “field value” group. For example, analysis result in case of query string is given as “name1=value1&name2=value2” as below.

```
char msg[] = “name1=value1&name2=value2”;
char *cgi_var[10];
char *cgi_val[10];
int cgi_cnt;
cgi_cnt = sizeof(cgi_var) / sizeof(char *);
CgiGetParamN( msg, strlen(msg), cgi_var, cgi_val, &cgi_cnt );

cgi_cnt = 2;
cgi_var[0] = “name1”;
cgi_var[1] = “name2”;
cgi_val[0] = “value1”;
cgi_val[1] = “value2”;
```

[Supplement]

The similar API, the CgiGetParam function, is left for compatibility. Use the CgiGetParam function can cause problems, so use the CgiGetParamN function unless you have a specific reason to do so.

CookieGetItem	Parsing cookie headers

【Format】

UB CookieGetItem(char **cookie, char **name, char **value)

【Parameter】

char	**cookie	Cookie header value
char	**name	Acquired cookie name
char	**value	The value of the acquired cookie name

【Return value】

UB	1 if cookie pair can be obtained, 0 otherwise

【Error code】

None

【Explanation】

This API gets the cookie name and value from the cookie header value ‘name1 = value1; name2 = value2; ..’ format. When the cookie pair can be obtained, the name is stored in name and the value is stored in value, and 1 is returned as the return value. Otherwise, the return value will be 0.

When this API is called, the pointer position pointed to by ** cookie will be the position of the next cookie pair, and the buffer contents will be changed. ** If you need the contents of the buffer pointed to by the cookie, save this API in another buffer before calling.

Example of use

```
void Http_Callback(T_HTTP_SERVER *http)
{
    VB *cname, *cval;
    VB buf[128];

    net_strcpy(buf, "<html><body>");

    /* Output the contents of the cookie header to HTML */
    net_strcat(buf, "\r\n<pre>\r\n");
    while (CookieGetItem(&http->hdr.cookie, &cname, &cval)) {
        net_strcat(buf, cname);    /* Cookie name */
        net_strcat(buf, "=");
        net_strcat(buf, cval);     /* Cookie name value */
        net_strcat(buf, "\r\n");
    }
    net_strcat(buf, "\r\n</pre>\r\n");
    net_strcat(buf, "</body></html>");
    HttpSendText(http, buf, net_strlen(buf));
}
```

HttpSendText	Transmission of text content

【Format】

ER ercd = HttpSendText(T_HTTP_SERVER *http, const char *str, UW len)

【Parameter】

T_HTTP_SERVER	*http	HTTP server control information
const char	*str	String to transmit
UW	len	length of string to transmit

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Wrong parameter is specified (*http is NULL.)

【Explanation】

This API transmits dynamic content. Please call this API only from CGI callback function.

Example

```
char page1[] = “<html><body>Welcome to this web server </body></html>”;

void Http_Callback(T_HTTP_SERVER *http)
{
   HttpSendText(http, page1, sizeof(page1));
}
```

HttpSendFile Send Attached File

【API】

```
ER ercd = HttpSendFile(T_HTTP_SERVER *http, const char *str,
                       UW len, const char *name, const char *type)
```

【Parameter】

T_HTTP_SERVER	*http	HTTP server information
const char	*str	File to be send
UW	len	Length to be send
const char	*name	Filename
const char	*type	Content-Type value or strings of HTTP header

【Return Value】

ER	ercd	Successful completion (E_OK) or error code

【Error Code】

E_PAR	Wrong parameter is specified (*http is NULL.)

【DESCRIPTION】

This API sends the dynamic content. Please only be called from this API function callback CGI.
Send file attachments in API: This is sent in (Content-Disposition attachment).

Use Case

```
char file[1024];

void Http_Callback(T_HTTP_SERVER *http)
{
int len;
                              :
/* Specify the "file" of contents, the size set to "len" */
                              :
   HttpSendFile(http, file , len, "FILE NAME", "text/plain");
}
```

HttpSendResponse	Send specified content

【API】

```
ER ercd = HttpSendResponse(T_HTTP_SERVER *http, const char *str,
                           UW len, const char *type)
```

【Parameter】

T_HTTP_SERVER	*http	HTTP server information
const char	*str	Byte string of content to transmit
UW	len	Byte length of content to transmit
const char	*type	Content type name

【Return Value】

ER	ercd	Successful completion (E_OK) or error code

【Error Code】

E_PAR	Wrong parameter is specified (*http is NULL.)

【DESCRIPTION】

This API sends the dynamic content. Please only be called from this API function callback CGI.

http_server_extcbk Callback function for extended operation

【API】

ER ercd = http_server_extcbk(T_HTTP_SERVER *http, const T_HTTP_FILE *fp, UB evt)

【Parameter】

T_HTTP_SERVER	*http	HTTP server information
T_HTTP_FILE	*fp	Accessed content information
UB	evt	Occurrence event judgment flag

【Return Value】

ER	ercd	Successful completion (E_OK) or error code

【DESCRIPTION】

Callback function (callback for extended operation) is called from the HTTP server task. When you access the content for which the extended operation flag is set from a Web browser, the callback function for extended operation is called. The accessed content information is entered in fp. In addition, the following values are entered in the judgment flag evt.

- HTTPD_EXT_AUTH (Use HTTP authentication)

HTTP authentication is applied to the relevant content. The callback function is called when the value of the Authorization header in the client request is in http-> hdr.auth or when http-> hdr.auth is NULL.

If http-> hdr.auth contains a value, check whether the user name / password (or digest value) of the value is correct in the function. As a result, design it to return E_OK on success and non-E_OK on failure.

If http-> hdr. Is NULL, WWW- written in the authentication failure response in the function

Create Authenticate header information (specify realm or nonce, etc.).

- HTTPD_EXT_UHDR (Specify custom header)

Callback function is called before replying the content information to the web browser, and any HTTP header can be added in the function. It does not apply to CGI.

(Applying this flag to CGI does not add a custom header)

.

Calling HTTP server APIs other than HttpSetContent () and APIs with waiting factors is prohibited in the callback function. The callback function uses a stack of HTTP server tasks.

Increase the stack size of the HTTP server task as needed. If there are multiple server tasks, the processing in the callback function must be reentrant.

Usage example and explanation are described below.

Use Case

```
【net_cfg.c - Content registration department】
/*******************************
    HTTP Content List
 *******************************/
T_HTTP_FILE const content_list[] =
{
    {"/", "text/html", index_html1, sizeof(index_html1), NULL, 0},
    {"/manage/", "text/html", manage_html2, sizeof(manage_html2), NULL, HTTPD_EXT_AUTH |
HTTPD_EXT_UHDR},
    {"/sample.cgi", "", NULL, 0, sample_fnc, HTTPD_EXT_AUTH},
    {"", NULL, NULL, 0, NULL}
};

[Net_cfg.c – net_setup () HTTP server task starter]
    {Omitted}

    gHTTP_EXT_CBK = httpd_evt_callback; / * Extended operation callback registration * /
    gHTTP_FILE = (T_HTTP_FILE *) content_list;

    / * Start HTTPd Task * /
    sta_tsk (ID_HTTPD_TSK1, 0);

    {Omitted}

[User source]
ER httpd_evt_callback (T_HTTP_SERVER * http, const T_HTTP_FILE * fp, UB evt)
{
    T_HTTP_HEADER * hdr = & http-> hdr;
    ER ercd;
    ercd = E_OK;

    switch (evt) {
    case HTTPD_EXT_AUTH: / * HTTP authentication (basic) * /
        if (hdr-> auth) {/ * with credentials * /
            {Omitted} / * Since the value is Base64, decode processing is performed * /
            {Omitted} / * Get username / password after decoding * /
            ercd = (OK == validation)? E_OK: E_OBJ;
        }
        else {/ * Authentication failed * /
            / * Create WWW-Authenticate header for response message * /
            HttpSetContent (http, "WWW-Authenticate: Basic");
            HttpSetContent (http, "realm = \" Secret Zone \ "\ r \ n");
        }
        break;
    case HTTPD_EXT_UHDR: / * Custom header specification * /
        HttpSetContent (http, "X-Frame-Options: DENY \ r \ n");
        break;
    }
    return ercd;
}
```

【DESCRIPTION】

Extended operation callback function is called when the following conditions are met.

-The processing callback function is registered with http_server_set_extcbk ().

-A value is set in the ext member of the content list.

-Access the above content from a web browser.

In the example, the callback function is not called when accessing http://xxx.xxx.xxx.xxx/, and the callback function is called when accessing other URLs.

Next, let's look at the processing inside the callback function. First, the callback function looks at the value of the flag evt to determine which extended operation (authentication processing, custom header processing) it is.

When the flag evt is HTTP authentication processing (HTTPD_EXT_AUTH), authentication check processing is performed when the value of hdr-> auth is other than NULL. The Authorization header value obtained from the browser is entered in hdr-> auth. The values are decoded and analyzed to check the validity of the user name, password, and digest value. (The user needs to handle BASE64 and MD5 processing.) If successful, specify E_OK, otherwise specify something other than E_OK as the return value.

If the authentication fails, the value of hdr-> auth is NULL and the callback is called. In this case, specify the WWW-Authenticate header information. (Up to line feed code)

If the flag evt is custom header processing (HTTPD_EXT_UHDR), arbitrary header information can be added to the end of the HTTP header prepared by the HTTP server task. Describe the header information that ends with a line feed code.

If need to determine the file name or content type, refer to fp.

HttpSetContent Addition of transmission buffer for HTTP server control information

【API】

```
ER ercd = HttpSetContent(T_HTTP_SERVER *http, const char *str);
```

【Parameter】

T_HTTP_SERVER	*http	HTTP server information
const char	*str	Clear the buffer by specifying the character string to be added and NULL

【Return Value】

ER	ercd	Number of additional bytes (≧ 0) or error code

【Error Code】

E_PAR	Wrong parameter is specified (*http is NULL.)

【DESCRIPTION】

This API adds a character string to the send buffer of HTTP server control information. Call this API only from the callback function for extended operation and the callback function for CGI.

When using it in the callback function for extended operation, do not specify NULL in the argument of str. The contents of the HTTP header created by the HTTP server application on the way will be cleared, and the operation will be undefined.

HttpSetContentKpa Add HTTP Keep-Alive header to send buffer

【API】

```
ER HttpSetContentKpa(T_HTTP_SERVER *http);
```

【Parameter】

T_HTTP_SERVER	*http	HTTP server information

【Return Value】

ER	ercd	Number of additional bytes (≧ 0) or error code

【Error Code】

E_PAR	Wrong parameter is specified (*http is NULL.)

【DESCRIPTION】

This API adds HTTP Keep-Alive header information to the send buffer of HTTP server control information. Call this API only from the CGI callback function.

HttpSetContentCookie Add Set-Cookie header to send buffer

【API】

```
ER HttpSetContentCookie(T_HTTP_SERVER *http,
                        const char *name, const char *val, const char *opt)
```

【Parameter】

T_HTTP_SERVER	*http	HTTP server information
const char	*name	Cookie name
const char	*val	Cookie name value
const char	*opt	Attribute specification buffer (optional)

【Return Value】

ER	ercd	Number of additional bytes (≧ 0) or error code

【Error Code】

E_PAR	Wrong parameter is specified (*http, *name, *val is NULL.)

【DESCRIPTION】

This API adds Set-Cookie header information to the send buffer of HTTP server control information. Call this API only from the callback function for extended operation and the callback function for CGI.

If need to specify the cookie attribute, specify the attribute in the string format in the argument opt. Specify the character string specified in opt in the format of {attribute name 1} = {value 1}. When specifying multiple attributes, use a semicolon (;) to combine the attribute pairs into a character string.

HttpSendBuffer	Send specified buffer contents

【API】

```
ER ercd = HttpSendBuffer(T_HTTP_SERVER *http, const char *str, UW len);
```

【Parameter】

T_HTTP_SERVER	*http	HTTP server information
const char	*str	Buffer to send
UW	len	Buffer length to send

【Return Value】

ER	ercd	Successful completion (E_OK) or error code

【Error Code】

E_PAR	Wrong parameter is specified (*http is NULL.)

【DESCRIPTION】

This API sends the send buffer of the HTTP server control information created by HttpSetContent () and the specified buffer. If NULL is specified for the argument str, only the data remaining in the send buffer of the HTTP server will be sent. Call this API only from the CGI callback function. The usage example is described below.

Use Case

```
ER HttpSendResponse (T_HTTP_SERVER * http, char * str, int len, char * type)
{
        / * Creating message header part * /
        HttpSetContent (http, 0);
        HttpSetContent (http, "HTTP / 1.1 200 OK \ r \ n");
        HttpSetContent (http, "Content-Type:");
        HttpSetContent (http, type);
        HttpSetContent (http, "\ r \ n");
        HttpSetContentLen (http, "Content-Length:", len);
        HttpSetContentKpa (http);
        HttpSetContent (http, "\ r \ n");

        / * Send message header and specified buffer (body) * /
        HttpSendBuffer (http, str, len);

        return E_OK;
}

/ * Callback function for HTTP server CGI (sample) * /
void user_cgi_callback (T_HTTP_SERVER * http)
{
        VB contents [128];
        net_strcpy (contents, "<html> <body> \ r \ n");
        net_strcat (contents, "<center> uNet CGI demo </ center> \ r \ n");
        net_strcat (contents, "</ body> </ html>");
        HttpSendResponse (http, contents, net_strlen (contents), "text / html");
}
```

6.3.2 HTTP server sample

```
/* Definition of content */
const char index_html[] =
"<html>\
<title> uNet3 HTTP Server </title>\
<body>\
<h1>Hello World!</h1>\
</body>\
</html>";

/* Initialization of content list */
T_HTTP_FILE const content_list[] =
{
    {"/", "text/html", index_html, sizeof(index_html), NULL},
    {"", NULL, NULL, 0, NULL} /* terminal */
};

/* Starting HTTP session */
static T_HTTP_SERVER http_server1;
void httpd_tsk1(VP_INT exinf)
{
/* Initialize the content list global pointer */
gHTTP_FILE = (T_HTTP_FILE*)content_list;

memset((char* )&http_server1, 0, sizeof(http_server1));
    http_server1.SocketID = ID_SOC_HTTP1;

    http_server(&http_server1);
}
```

6.4 DNS client

In DNS client, use UDP socket. UDP socket will be created by the below parameter.

ID	protocol	port	timeout	
			send	receive
ID_SOC_DNS	UDP	0	5s	5s

(1) DNS Client Information

```
typedef struct t_dns_client {
        UW              ipa;            / * DNS server IP address * /
        char            *name;          / * Host name (for setting / reference) * /
        UW              *ipaddr;        / * IP address (for setting / reference) * /
        SID             sid;            / * DNS Socket ID (UDP) * /
        UH              code:           / * Request RR type * /
        UB              dev_num;        / * Device number * /
        UB              retry_cnt;      /* number of retries */
} T_DNS_CLIENT ;
```

Use this structure as an argument to the dns_query_ext () API. Be sure to specify ipa, name, ipaddr, and sid. If not specified, E_PAR will be returned. You can specify the request type with code. At the moment, only the following definition values can be specified. Otherwise, E_NOSPT will be returned.

The device number (dev_num) and the number of retries (retry_cnt) can be specified as required. If the number of retries is not specified (0), it will not be retried. Retrying will perform retransmission processing when a reception timeout occurs.

- DNS client RR definition value

```
#define RR_TYPE_A          1U      Get an IP address
#define RR_TYPE_PTR        12U     Get the host name
#define RR_TYPE_AAAA       28U     Get an IPv6 address
                                   (Separately, eForce TCP / IP stack μNet3 / IPv6 is
                                   required)
```

Defined value in T_DNS_CLIENT :: code

6.4.1 DNS client API

dns_get_ipaddr	Acquire IP address from host name

【Format】

```
ER ercd = dns_get_ipaddr(SID socid, UW dns_server, char *name, UW *ipaddr);
```

【Parameter】

SID	socid	UDP socket ID
UW	dns_server	IP address of DNS server
char	*name	Host name
UW	*ipaddr	IP address to acquire

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Wrong parameter is specified
E_TMOUT	No response from DNS server
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	Unable to resolve IP address from host name

Example of use

```
UW ip;
ER ercd;
UW dns_server = ip_aton(“192.168.11.1”);

dns_get_ipaddr(ID_SOC_DNS, dns_server, “TEST Server URL”, &ip);
```

dns_get_name Acquire host name from IP address

【Format】

```
ER ercd = dns_get_name(SID socid, UW dns_server, char *name, UW *ipaddr);
```

【Parameter】

SID	socid	UDP socket ID
UW	dns_server	IP address of DNS
char	*name	host name to acquire
UW	*ipaddr	IP address

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Wrong parameter is specified
E_TMOUT	No response from DNS server
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	Unable to acquire host name from IP address

Example of use

```
UW ip = ip_aton(“192.168.11.30”);
ER ercd;
char host_name[256];
UW dns_server = ip_aton(“192.168.11.1”);

dns_get_name(ID_SOC_DNS, dns_server, host_name, &ip);
```

dns_query_ext	Issuing DNS queries

【Format】

ER ercd = dns_query_ext(T_DNS_CLIENT *dc);

【Parameter】

T_DNS_CLIENT	*dc	DNS client information

【Return value】

ER	ercd	Successful completion (E_OK) or error code

【Error code】

E_PAR	Wrong parameter is specified
E_TMOUT	No response from DNS server
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	Unable to acquire host name from IP address
E_NOSPT	An unsupported RR was specified.

【DESCRIPTION】

An extended version API of dns_get_ipaddr () and dns_get_name (). Compared to the API on the left, it is now possible to specify a new device number (dc-> dev_num) to be sent and retry (dc-> retry_cnt) when a failure occurs.

Example of use

```
ER ercd;
T_DNS_CLIENT dc = {0};
char host_name[256];
UW dns_server = ip_aton("192.168.11.1");

dc.code       = RR_TYPE_PTR;
dc.name       = name;
dc.ipaddr     = ip_aton("192.168.11.30");
dc.ipa        = dns_server;
dc.sid        = ID_SOC_DNS;
//dc.dev_num   = 0;
//dc.retry_cnt = 1;

ercd = dns_query_ext(&dc);
```

6.5 DHCP client extended

For the existing DHCP client, holds the lease on the resources of the state, such as IP, DHCP client extended version information (RENEW), release (RELEASE), denial (DECLINE), a restart (REBOOT), an extension of these has been enhanced to provide the function get (INFORM).

(1) DHCP Client Information

```
typedef struct t_dhcp_client {
        T_DHCP_CTL      ctl             /* Control Informatio */
        UW              ipaddr;         /* IPaddress*/
        UW              subnet;         /* Subnet Mask */
        UW              gateway;        /* Gateway Address */
        UW              dhcp;           /* DHCPserver address */
        UW              dns[2];         /* DNSAddress */
        UW              lease;          /* Release time for DHCP Addless */
        UW              t1;             /* Renewal DHCP address */
        UW              t2;             /* Rebind time of DHCP address.*/
        UB              mac[6];         /* MAC addresss */
        UH              dev_num;        /* Device number */
        UB              state           /* Status of DHCP Clients */
        SID             socid;          /* UDP with socket ID */
        UB              arpchk;         /* Duplicate IP check */
        T_DHCP_UOPT     *uopt;          / * DHCP option acquisition parameters * /
        UB              uopt_len;       / * Number of DHCP option acquisition parameters * /
        UB              retry_cnt;      /* number of retries */
} T_DHCP_CLIENT ;
```

This structure is intended to be used as an argument to the DHCP client API, is an extension of the host address information structure. The same manner as described above, UDP socket ID number and device must be set by the user application. Please refer to the DHCP client is the value to be set.

If you set the "ARP_CHECK_ON" to check whether or not duplicate IP, you do the duplicate check using the ACD feature for the IP, which is leased from the DHCP server.

If want to acquire arbitrary option information (*) from the DHCP server, specify the DHCP option acquisition parameter. If you specify a value, the specified option code is added to the Parameter Request List of the DHCP DISCOVER / REQUEST / INFORMATION message, and an attempt is made to get the option from the response message. Specify 0 if it is not required.

* The value format that can be acquired is either binary (1,2,4 bytes), character string, or address.

It is possible to retry the request if there is no response. If the number of retries (retry_cnt) is not specified (0), retries will be performed 3 times. Please specify if necessary.

This structure is used, even when I update the IP address that you set IP address at the time of acquisition. We cannot change the DHCP client state and control information in the application for that.

- DHCP client status definition value

```
#define DHCP_STS_INIT           0
#define DHCP_STS_INITREBOOT     1
#define DHCP_STS_REBOOTING      2
#define DHCP_STS_REQUESTING     3
#define DHCP_STS_BOUND          4
#define DHCP_STS_SELECTING      5
#define DHCP_STS_REBINDING      6
#define DHCP_STS_RENEWING       7
```

Defined value in T_DHCP_CLIENT :: state

(2) DHCP acquisition option information

```
typedef struct t_dhcp_uopt {
        UB          code;       / * DHCP option code * /
        UB          len;        / * Optional element size / piece * /
        UB          ary;        / * Number of optional elements * /
        UB          flag;       / * Element discrimination / status flag * /
        VP          val;        / * Element storage destination pointer * /
} T_DHCP_UOPT;
```

This structure is used to get arbitrary optional information from the DHCP server. Code number of the option you want to get in code, size per value of the target option in len, number of elements of the target option in ary, flag value setting described later in flag, variable where the value is actually stored in val Specify each array. An example of setting options is shown below.

	Time Offset(2)	Host Name(12)	Log Server(7)
code;	2	12	7
len;	4	1	4
ary;	1	Buffer size	1… n (when multiple acquisitions)
flag;	DHCP_UOPT_BIN	DHCP_UOPT_STR	DHCP_UOPT_IPA
val;	NT type variable point	Buffer pointer	UW type variable / array pointer

After specifying the value of this structure in the DHCP option acquisition parameter of the T_DHCP_CLIENT structure, call either dhcp_bind () or dhcp_inform (). If the response message contains the target option, the acquired value is entered at the pointer of val. If the value can be obtained, the DHCP_UOPT_STS_SET bit value is set in flag.

Next, the actual usage of the T_DHCP_UOPT structure is described.

6.5.1 DHCP client extended API

dhcp_bind	Get DHCP Lease Information

【API】

```
ER ercd = dhcp_bind(T_DHCP_CLIENT *dhcp);
```

【Parameter】

T_DHCP_CLIENT	*dhcp	DHCP Client Information

【Return Value】

ER	ercd	Success (E_OK) or Error Code

【Error Code】

E_PAR	*dhcp is NULL, or social is not specified
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	Incorrect Socket status (No create the socket)
E_SYS	Address conflict another host when the IP address is assigned.
E_TMOUT	Response is delay from DHCP server or the DHCP server doesn't exist.

【DESCRIPTION】

This API provides the same functionality as the () API dhcp_client traditional.

To verify the IP address that you get that you do not have overlap with other hosts, we set up a check for duplicate IP ARP_CHECK_ON the presence of DHCP client information of the argument. If a duplicate IP address is detected at this time, to send a message to the DHCP server DHCP_DECLINE, API will return the E_SYS.

dhcp_renew	Renewal DHCP Lease

【API】

```
ER ercd = dhcp_renew(T_DHCP_CLIENT *dhcp);
```

【Parameter】

T_DHCP_CLIENT	*dhcp	DHCP Client Information

【Return Value】

ER	ercd	Success(E_OK) or Error Code

【Error Code】

E_PAR	*dhcp is NULL, or social is not specified
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	Incorrect DHCP client, or request denied by DHCP server.
E_SYS	Address conflict another host when the IP address is assigned.
E_TMOUT	Response is delay from DHCP server or the DHCP server doesn't exist.

【DESCRIPTION】

This API to extend the validity period of IP address obtained from the DHCP server. The argument specifies the DHCP client information obtained by dhcp_bind ().

This API should be called (t_1) within the validity period. Lifetime is measured by the application using the timer or control task.

This feature also includes the ability RENEW REBIND. The difference between the two is only to send broadcast messages to send unicast REQUEST. If you cannot receive the ACK after sending a REQUEST message to the DHCP server at the beginning, we perform a broadcast transmission immediately.

To verify the IP address that the extension does not have an overlap with other hosts, we set up a check for duplicate IP ARP_CHECK_ON the presence of DHCP client information of the argument. If a duplicate IP address is detected at this time, to send a message to the DHCP server DHCP_DECLINE, API will return the E_SYS.

Only the RENEW function is performed. If you want to make the operation the same as the R-IN32M3 TCP/IP stack, please call dhcp_rebind () after checking the error of this API.

dhcp_rebind Application for reassignment of lease information

【API】

```
ER ercd = dhcp_rebind(T_DHCP_CLIENT *dhcp);
```

【Parameter】

T_DHCP_CLIENT	*dhcp	DHCP Client Information

【Return Value】

ER	ercd	Success(E_OK) or, Error Code

【Error Code】

E_PAR	*dhcp is NULL, or social is not specified
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	Illegal DHCP client information or DHCP server has refused the request.
E_SYS	The assigned IP address conflict with another host.
E_TMOUT	Delayed response from the DHCP server, or DHCP server does not exist.

【DESCRIPTION】

This API requests reassignment of the IP address obtained from the DHCP server. For the argument, specify the DHCP client information obtained by dhcp_bind ().

Normally, call this API after T2 hours have passed. The validity period is measured by the application using a timer and task control.

To verify that the extended IP address is not duplicated with other hosts, set ARP_CHECK_ON for the presence or absence of IP duplicate check in the DHCP client information argument. If duplicate IP addresses are detected at this time, a DHCP_DECLINE message is sent to the DHCP server, and the API returns E_SYS.

dhcp_reboot DHCP Client Reboot

【API】

```
ER ercd = dhcp_reboot(T_DHCP_CLIENT *dhcp);
```

【Parameter】

T_DHCP_CLIENT	*dhcp	DHCP Client Information

【Return Value】

ER	ercd	Success(E_OK) or, Error Code

【Error Code】

E_PAR	*dhcp is NULL, or social is not specified, or there is no address available.
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	Illegal DHCP client information or DHCP server has refused the request.
E_SYS	The assigned IP address conflict with another host.
E_TMOUT	Delayed response from the DHCP server., or DHCP server does not exist.

【DESCRIPTION】

This API is reusing the IP resource to which the client was previously used, API is used to verify its legitimacy in DHCP server. If you do not, such as when you remove and insert the LAN cable or if the LAN interfaces in the pause is activated again, and security that are part of the same network before and after, the DHCP server for the IP resource that has been previously used, for example notice.

The argument specifies the DHCP CLIENT INFORMATION dhcp_bind acquired by ().

If an ACK is not received after sending REQUEST message, API This is an error, or if it receives a DHCPNAK.

To verify the IP address that you notice that you do not have overlap with other hosts, we set the duplicate IP ARP_CHECK_ON to check whether or not the argument of the DHCP CLIENT INFORMATION. If a duplicate IP address is detected at this time, to send a message to the DHCP server DHCP_DECLINE, API will return the E_SYS.

dhcp_release Release DHCP Lease Information

【API】

```
ER ercd = dhcp_release(T_DHCP_CLIENT *dhcp);
```

【Parameter】

T_DHCP_CLIENT	*dhcp	DHCP CLIENT INFORMATION

【Return Value】

ER	ercd	Success (E_OK) or Error Code

【Error Code】

E_PAR	*dhcp is NULL, or social is not specified
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	Illegal DHCP client information
E_TMOUT	Timeout send DHCPRELEASE message

【DESCRIPTION】

This API will notify the DHCP server to release the resources when it no longer want to use the IP address obtained from the DHCP server.

The argument specifies the DHCP information obtained dhcp_bind().

dhcp_inform Get DHCP Options

【API】

```
ER ercd = dhcp_inform(T_DHCP_CLIENT *dhcp);
```

【Parameter】

T_DHCP_CLIENT	*dhcp	DHCP CLIENT INFORMATION

【Return Value】

ER	ercd	Success (E_OK) or Error Code

【Error Code】

E_PAR	*dhcp is NULL, or social is not specified, or there isn’t available of the address.
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	No IP address has been set for the host.
E_TMOUT	Delay DHCP server response, or no existence DHCP server.

【DESCRIPTION】

This API to get the information other than the IP address from the DHCP server. Set the static IP address, for example, the address of the DNS server is used, for example, if you want to get from the DHCP server.

To the argument set DHCP CLIENT INFORMATION (for TCP/IP stack) only socket ID and device number of the interface.

6.5.2 DHCP client extended information

DHCP Client Extended Information

```
ER ercd;
        T_DHCP_CLIENT dhcp;
        T_DHCP_UOPT dhcp_uopt [3];
        INT uo_i4;
        UINT uo_u4;
        VB uo_buf [64];

        / * Setting the T_DHCP_UOPT structure * /
        net_memset (& dhcp_uopt [0], 0, sizeof (dhcp_uopt));

        / * Time offset (2), Host Name (12), Log Server (7) * /
        SET_DHCP_UOPT_BIN4 (dhcp_uopt [0], 2, & uo_i4);
        SET_DHCP_UOPT_STR (dhcp_uopt [1], 12, uo_buf, sizeof (uo_buf));
        SET_DHCP_UOPTS_IPA (dhcp_uopt [2], 7, uo_u4, sizeof (uo_u4));
        / * {T_DHCP_CLIENT structure settings} * /
        dhcp.uopt = dhcp_uopt;
        dhcp.uopt_len = 3;

        / * {... T_DHCP_CLIENT structure settings ...} * /
        ercd = dhcp_bind (& dhcp);
        if (E_OK == ercd) {
              for (ercd = 0; ercd <sizeof (dhcp_uopt); ++ ercd) {
                    if (dhcp_uopt [ercd] .flag & DHCP_UOPT_STS_SET) {
                          / * Performs processing when options are acquired * /
                          / * dhcp_uopt [ercd] .val pointer has a value * /
                    }
              }
        }
```

Source commentary

In the above example, it is set to get the options of TimeOffset (2), HostName (12), LogServer (7). Macros are used to simplify the process. Note that you need a variable to store the value of the target option, separate from the variables in the T_DHCP_UOPT structure. The value of the option acquired after executing the DHCP client API for information acquisition (here, dhcp_bind ()) is stored in the above variable.

- T_DHCP_UOPT element discrimination / status flag definition

```
/ * User set value * /
#define DHCP_UOPT_STR            0x80       Options are in string format
#define DHCP_UOPT_IPA            0x40       Options are address format
#define DHCP_UOPT_BIN            0x20       Options are in binary format
/ * User reference value * /
#define DHCP_UOPT_STS_SET        0x01       Option value set
```

Defined value in T_DHCP_UOPT :: flag

- T_DHCP_UOPT setting macro

```
/ * For when the acquisition value of the option is singular * /
SET_DHCP_UOPT_BIN1(_uopt_,_code_,_pval_)              For 1 byte binary
SET_DHCP_UOPT_BIN2(_uopt_,_code_,_pval_)              For 2-byte binaries
SET_DHCP_UOPT_BIN4(_uopt_,_code_,_pval_)              For 4-byte binaries
SET_DHCP_UOPT_IPA(_uopt_,_code_,_pval_)               For address
SET_DHCP_UOPT_STR(_uopt_,_code_,_pval_,_len_)         For strings


/ * For multiple option acquisition values * /
SET_DHCP_UOPTS_BIN1(_uopt_,_code_,_pval_,_len_)       For 1 byte binary
SET_DHCP_UOPTS_BIN2(_uopt_,_code_,_pval_,_len_)       For 2-byte binaries
SET_DHCP_UOPTS_BIN4(_uopt_,_code_,_pval_,_len_)       For 4-byte binaries
SET_DHCP_UOPTS_IPA(_uopt_,_code_,_pval_,_len_)        For address
```

Auxiliary macro for setting the value of the T_DHCP_UOPT structure. Specify the following as macro arguments.
uopt: The entity of the T_DHCP_UOPT structure, _code_: Option code,
pval: Pointer for storing option values, _len_: Number of elements that can be acquired (number of elements of _pval_)

6.6 Ping Client

Ping client sends an ICMP echo request to any destination. If there is an echo response from the other party, you can see that communication with the IP address is possible.

(1) PING client control information

```
typedef struct t_ping_client {
        SID             sid;            / * ICMP socket ID * /
        UW              ipa;            / * Destination IP address * /
        TMO             tmo;            / * Waiting for response timeout (ms) * /
        UH              devnum;         / * Device number * /
        UH              len;            / * Packet size (bytes) * /
} T_PING_CLIENT ;
```

Set the required information in this structure and pass it as an argument of the PING client API.

The Ping client uses the ICMP socket to send and receive.
Create a socket with cre_soc as follows: Set the port number to 0. Assign the ID of the created socket to the sid of T_PING_CLIENT of the structure.

Create ICMP socket

```
T_NODE node;

    node .num = 1; / * Network device number * /
    node .ipa = INADDR_ANY;
    node .port = 0; / * Port should 0 for ICMP * /
    node .ver = IP_VER4;
    sid = soc_cre (IP_PROTO_ICMP, & node);
    if (sid <= 0) {
        return E_NOMEM;
    }
```

6.6.1 Ping Client API

ping_client	ICMP Echo (Transmite Request and Receive Response)

【API】

```
ER ping_client(T_PING_CLIENT *ping_client);
```

【Parameter】

T_PING_CLIENT	*ping_client	Ping transmit information

【Return Value】

ER	ercd	Success (E_OK) or Error Code

【Error Code】

E_PAR	Specified the incorrect Parameter
E_TMOUT	No response from remote or failed address resolver
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	Other errors

【DESCRIPTION】

Ping the destination IP address. This function keeps waiting until a response is returned from the destination. If there is no response from the destination, a timeout (E_TMOUT) error is returned.

6.7 SNTP Client

SNTP client get the (number of seconds starting from 1/1/1900) time from NTP (NTP) server time on the network using the NTP packet.

(1) SNTP client control information

```
typedef struct t_sntp_client {
        SID             sid;            /* Socket ID */
        UW              ipa;            /* SNTP server IP address */
        TMO             tmo;            /* Timeout setting */
        UH              devnum;         /* Device number */
        UH              port;           /* port number */
        UB              ipv;            /* IP version */
        UB              stt;            /* Received data Stratum field value */
} T_SNTP_CLIENT ;
```

Set the required information in this structure and pass it as an argument of the SNTP client API.

- Device number

For the device number, specify the network device used by the SNTP client. If ‘0’ is specified, the default network device will be used. (Normally, set 0.)

- SNTP server IP address

Specify the IP address of the connection destination SNTP server.

- Time-out setting

Specify the timeout period for receiving the response packet from the SNTP server.

- port number

Specify when changing the connection destination port number of the SNTP server. Normally, enter 0.

- IP version

Specify the IP version.

- Received data Stratum field value

If sntp_client () completes normally, the value of the Stratum field of the received data will be entered.
The Stratum field usually has a value between 0 and 15, where 0 is time asynchronous or time source unknown and 1 to is the clock hierarchy level. (1 is the route hierarchy) If there is a server where the value of the Stratum field is 0 until the correct time response, judge with this value.

6.7.1 SNTP Client API

sntp_client	Get NTP time

【API】

```
ER ercd = sntp_client(T_SNTP_CLIENT *sntp_client, UW *sec, UW *fra);
```

【Parameter】

T_SNTP_CLIENT	*sntp_client	Information of SNTP client
UW	*sec	NTP Time (second)
UW	*fra	NTP time (fractional seconds)

【Return Value】

ER	ercd	Success (E_OK) or Error Code

【Error Code】

E_PAR	Specified the illegal parameter
E_TMOUT	No response from remote or failed address resplver
E_NOMEM	Insufficient memory (exhausted network buffer)
E_OBJ	Incorrect Information of SNTP client

【DESCRIPTION】

This API gets the NTP time from SNTP server you set up in the argument. To set the SNTP server, specify the IPv4 address and port number.

In the SNTP client uses the UDP socket. If TCP/IP stack argument should be set to the socket ID available.

This API returns the E_OK if you can successfully get the time NTP.

NTP time is shown in the sec and fra arguments at this time. Because you are starting from 1/1/1900, NTP time and the conversion to Unix time (JST) UTC must be calculated by the caller.

Use Case

```
T_SNTP_CLIENT sc = {0};
    UW sec, fra;
    ER ercd;

    sc.sid = ID_SOC_SNTPC;
    ercd = sntp_client (& sc, & sec, & fra);
    if (ercd == E_OK) {
        / * UnixTime conversion * /
        sec-= 2208988800;

        / * Millisecond precision integer representation * /
        fra = ((fra >> 16) * 1000) >> 16;
    }
```

6.8 String Library

TCP/IP stack system provides a standard library of String so that it is not dependent on the compiler. Network applications you can use to provide these functions.

net_atoi	Numerical conversion of character string to int type

【API】

int net_atoi(const char *str);

【Parameter】

const char	*str	Target character string

【Return Value】

int	Result

【DESCRIPTION】

Converts the first part of the string specified by str to an int type integer. Returns 0 if conversion is not possible.

net_atol	Numerical conversion of character string to long type

【API】

long net_atol(const char *str);

【Parameter】

const char	*str	Target character string

【Return Value】

Long	Result

【DESCRIPTION】

Converts the first part of the string specified by str to a long integer. Returns 0 if conversion is not possible.

net_itoa String conversion of int type number

【API】

```
char* net_itoa(int num, char *str, int base);
```

【Parameter】

int	num	Target numerical value
char	*str	Conversion result string
int	base	Conversion radix

【Return Value】

char *	Conversion result string

【DESCRIPTION】

A non-standard C library. Converts the number indicated by num to a string with the radix of base and returns the result in str and the return value.

net_strncasecmp Compare String (case-insensitive letter)

【API】

```
int net_strncasecmp(const char *str1, const char *str2, SIZE len);
```

【Parameter】

const char	*str1	String to be compared
const char	*str2	String to be compared
SIZE	len	Length of compare

【Return Value】

int	Result

【DESCRIPTION】

The results were compared with the character code, it will return 0 if str1 = str2. The return value is positive if str1> str2, and is negative if str1 <str2.

I arrived at the end of either string number of characters until it reaches the comparison is to be compared. By this function equate the case of the letters.

net_strcmp String Compare

【API】

```
int net_strcmp(const char *str1, const char *str2);
```

【Parameter】

const char	*str1	String to be compared
const char	*str2	String to be compared

【Return Value】

int	Result of Compare

【DESCRIPTION】

The results were compared with the character code, it will return 0 if str1 = str2. The return value is positive if str1> str2, and is negative if str1 <str2.

Reach the end of the string until one of them to be compared

net_strncmp String Compare

【API】

```
int net_strncmp(const char *str1, const char *str2, SIZE len);
```

【Parameter】

const char	*str1	String to be compared
const char	*str2	String to be compared
SIZE	len	Length of compare

【Return Value】

int	Result of Compare

【DESCRIPTION】

The results were compared with the character code, it will return 0 if str1 = str2. The return value is positive if str1> str2, and is negative if str1 <str2. Comparison is made for the number of comparison characters or until the end of any character string is reached.

net_strcpy	String Copy

【API】

```
char* net_strcpy(char *str1, const char *str2);
```

【Parameter】

char	*str1	Address of copy destination string
const char	*str2	Address of copy source string

【Return Value】

char *	Address of copy destination string

【DESCRIPTION】

This API is to copy of the srt2 to the end of str1 (NULL).

net_strlen	Get String Length

【API】

```
SIZE net_strlen(const char *str);
```

【Parameter】

char	*str	String

【Return Value】

SIZE	String length

【DESCRIPTION】

Gets the number of characters up to (NULL) end of str. (NULL is not included)

net_strncat String concatenation

【API】

```
char* net_strncat(char *str1, const char *str2, SIZE len);
```

【Parameter】

char	*str1	Address of destination string
const char	*str2	Address of source string
SIZE	len	Length of string

【Return Value】

char *	Address of destination string

【DESCRIPTION】

Copies up to the number of concatenated characters of str2 or the end, starting from the end (NULL) of the concatenation character string str1.

net_strcat String concatenation

【API】

```
char* net_strcat(char *str1, const char *str2);
```

【Parameter】

char	*str1	Address of destination string
const char	*str2	Address of source string

【Return Value】

char *	Address of destination string

【DESCRIPTION】

Copy to the end of the str2 starting at (NULL) coupling the end of the destination string str1.

net_strchr Search Character

【API】

```
char* net_strchr(const char *str, int ch);
```

【Parameter】

const char	*str	Search target string
int	ch	Search character

【Return Value】

char *	Address where the search string of the search target string appears

【DESCRIPTION】

Searches whether the search character ch exists from the beginning to the end (NULL) of the search target character string str. If the search character exists, its start address is returned, and if it does not exist, NULL is returned as the return value.

net_strstr Search String

【API】

```
char* net_strstr(const char *str1, const char *str2);
```

【Parameter】

const char	*str1	Search target string
const char	*str2	Search string

【Return Value】

char *	Address where the search string of the search target string appears

【DESCRIPTION】

Searches if the search string str2 exists from the beginning to the end (NULL) of the search target string str1. If the search string exists, its start address is returned, and if it does not exist, NULL is returned as the return value.

net_strcasestr Character string search: Uppercase and lowercase letters are identified

【API】

```
char* net_strcasestr(const char *str1, const char *str2);
```

【Parameter】

const char	*str1	Search target string
const char	*str2	Search string

【Return Value】

char *	Address where the search string of the search target string appears

【DESCRIPTION】

Searches if the search string str2 exists from the beginning to the end (NULL) of the search target string str1. If the search string exists, its start address is returned, and if it does not exist, NULL is returned as the return value.

This function equates uppercase and lowercase letters.

net_strncpy N-character copy of a string

【API】

```
char* net_strncpy(char *str1, const char *str2, SIZE len);
```

【Parameter】

char	*str1	Address of copy destination string
const char	*str2	Address of copy source string
SIZE	len	Number of copy characters

【Return Value】

char *	Address of copy destination string

【DESCRIPTION】

Copies up to the end (NULL) or len (number of copy characters) of the copy source character string str2 to str1.

If len is less than the length of str2, only the len character is copied and no terminating character is added to the len + 1 character of str1.

If len is greater than the length of str2, the string after the end of str2 is padded with NULL up to the length of len.

7. Tutorial by sample application

In this chapter, the way to run the TCP/IP stack sample application is showed, and the behavior of it is confirmed.

7.1 Descriptions of sample software

"uNet3_sample" under the directory "Project" is used. This sample software lets us confirm HTTP server.

- Web server
  LED flashing period can be changed by 100 msec unit from the Web browser.

Please refer to *2.3 Directory and file organization* for a list of files included in sample software. And please note that R-IN32 TCP/IP stack is need to merge to driver and middleware sample program.

7.2 Hardware connection

Please check file DDR_ETH_CFG.h.
If macro USE_ETHSW is set to 0, please use the R-IN32M4-CL3 board's RJ45 **PORT0**. See following figure.

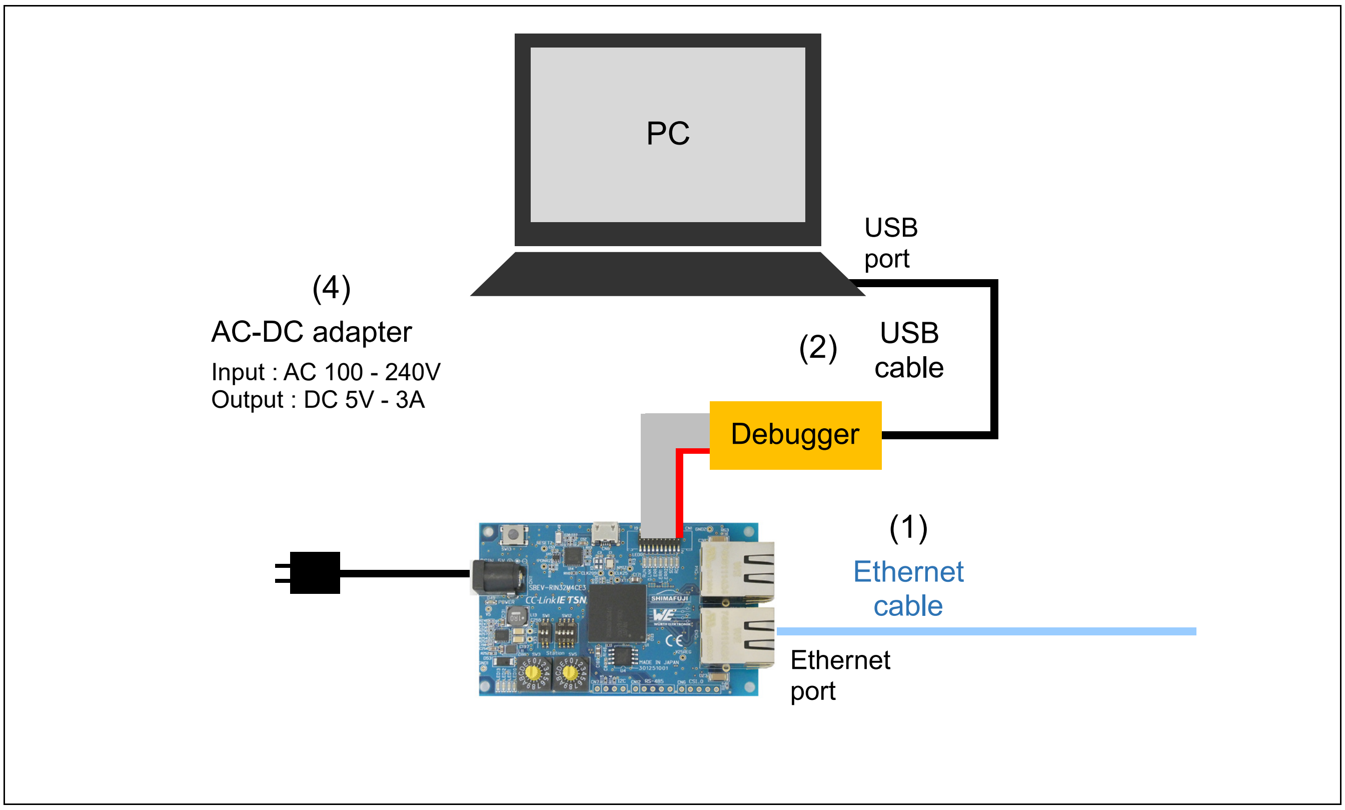


Fig. 7.1 Physical port connection for sample application

7.3 Board IP address setting

There are two options for setting the IP address. Either use a fixed IP address, or let the LAN network DHCP controller set it.

7.3.1 Setting for use a fixed IP address

Please set according to following procedure.

(1) Set DHCP_ENA to 0 in net_sample.c.

(2) Set desired server network address setting in net_cfg.c. Example is shown in Fig. 7.2.

```
/*******************************************
      Define Local IP Address
*******************************************/
T_NET_ADR gNET_ADR[] = {
      {
            0x0,                  /* Reserved */
            0x0,                  /* Reserved */
            0xC0A80164,         /* IP address (192.168.   1.100) */
            0xC0A80101,         /* Gateway       (192.168.   1.   1) */
            0xFFFFFF00,         /* Subnet mask (255.255.255.   0) */
      }
};
```

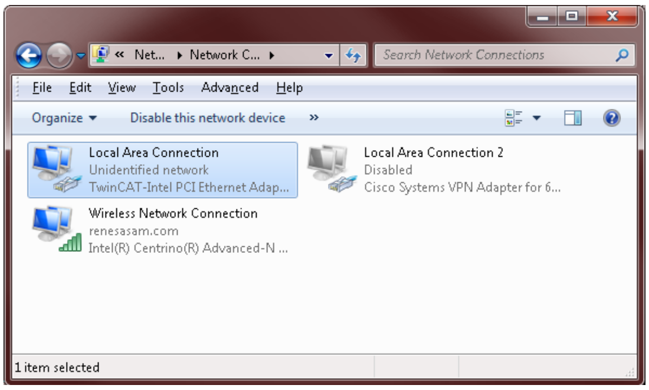
Fig. 7.2 The example setting of IP address (in case IP address is 192.168.1.100)

(3) Your PC's IP-address need to be in the same domain as the R-IN32 board. (Please also refer next page as detail procedure.)
In this example we will use:
Subnet mask: 255.255.255.0.
PC IP-address: 192.168.1.101.
This is so that server and client are in the same domain.

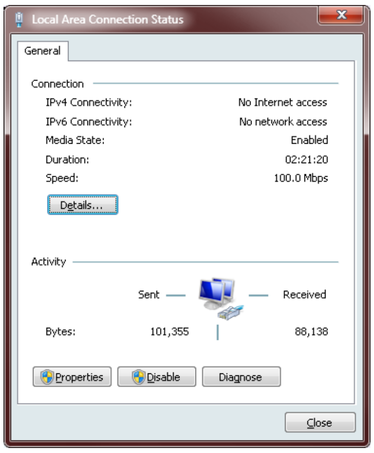
(4) Skip next section "7.3.2 Setting for use DHCP ".

cf. How to set the PC IP-address (not using DHCP)

- Open the network connections list.
  In Windows7: Control panel->Network and Sharing Center->Change adapter settings.


Net... ▸ Network C... ▸
Search Network Connections
File Edit View Tools Advanced Help
Organize ▾ Disable this network device »
Local Area Connection
Unidentified network
TwinCAT-Intel PCI Ethernet Adap...
Local Area Connection 2
Disabled
Cisco Systems VPN Adapter for 6...
Wireless Network Connection
renesasam.com
Intel(R) Centrino(R) Advanced-N ...
1 item selected

- Double-click (or right-click) on the Local Area Connection, then select "Properties".

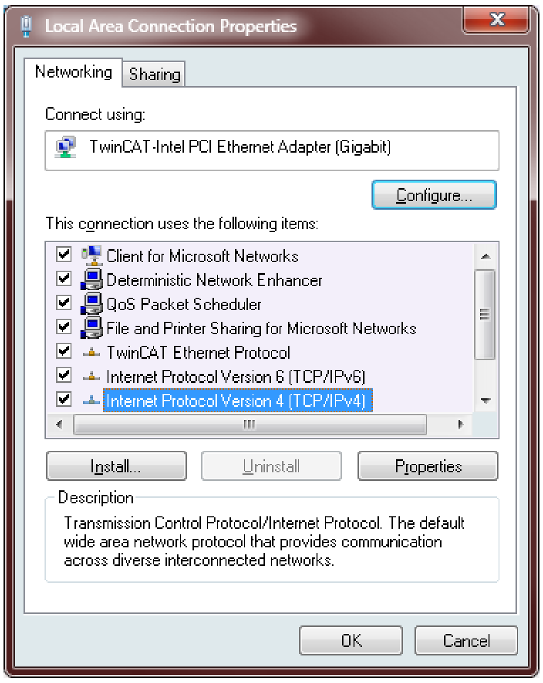

Local Area Connection Status

General

Connection
IPv4 Connectivity: No Internet access
IPv6 Connectivity: No network access
Media State: Enabled
Duration: 02:21:20
Speed: 100.0 Mbps

Activity
Sent — Received
Bytes: 101,355 88,138

- Select TCP/IPv4, and push the Properties button.


Local Area Connection Properties

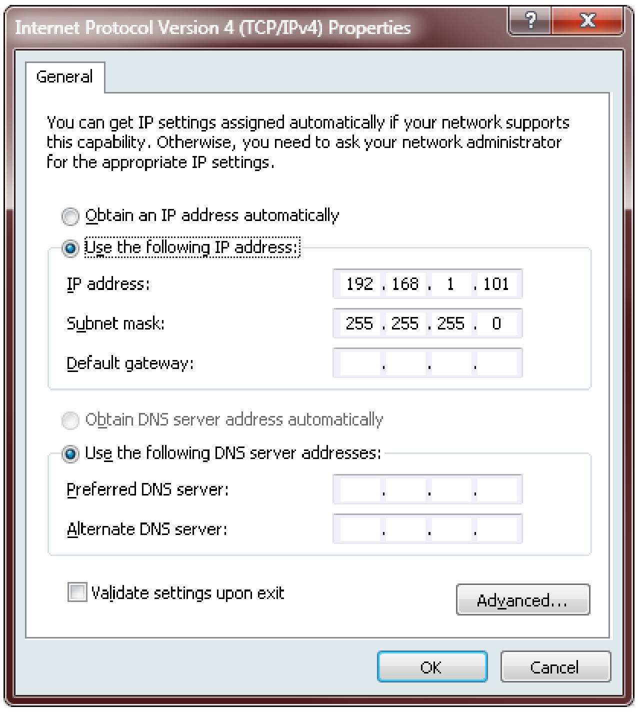
Networking | Sharing

Connect using:
TwinCAT-Intel PCI Ethernet Adapter (Gigabit)

This connection uses the following items:
- [✓] Client for Microsoft Networks
- [✓] Deterministic Network Enhancer
- [✓] QoS Packet Scheduler
- [✓] File and Printer Sharing for Microsoft Networks
- [✓] TwinCAT Ethernet Protocol
- [✓] Internet Protocol Version 6 (TCP/IPv6)
- [✓] Internet Protocol Version 4 (TCP/IPv4)

Description
Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

- Set IP-address to 192.168.1.101, and sub net mask to 255.255.255.0



You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address:	192 . 168 . 1 . 101
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	. . .

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

Preferred DNS server:	. . .
Alternate DNS server:	. . .

Validate settings upon exit

Done.

7.3.2 Setting for use DHCP function

When DHCP client is enabled, IP address is defined automatically by DHCP server. In this case, UDP socket for DHCP is also added automatically.

Please set according to following procedure.

(1) Set DHCP_ENA to 1 in net_sample.c.
(2) Connect the LAN cable to port 1
(3) Connect the LAN cable to the PC.

Note If user uses EWARM evaluation version which is size limited 32KB and set DHCP_ENA to 1, the sample project might not be able to compile because of size limitation.

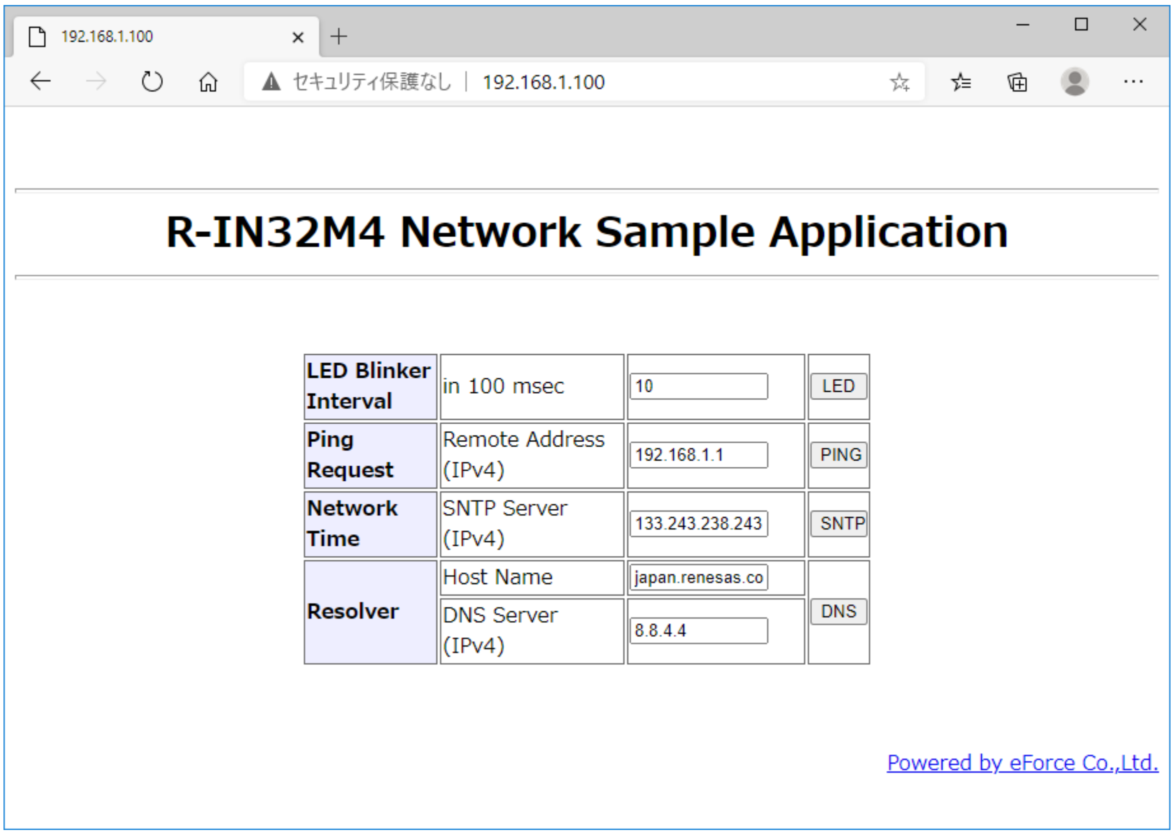
7.4 Demonstration

7.4.1 Webserver

In this example we will use the IAR project “uNet3_sample”.
Please execute the following procedure.

1. Compile, download, and run application.
2. Open a web browser in PC (client),
3. Enter into the URL field http://192.168.1.100 (or http://192.168.1.100:80 , but socket is default 80 for http).

If the board is running the project above, you should see a webpage from the R-IN32 like Fig. 7.3 .
If you have problems getting or seeing anything in the browser, you may need to restart the code in IAR, and/or get rid of breakpoints that cause timeouts.



R-IN32M4 Network Sample Application

LED Blinker Interval	in 100 msec	10	LED
Ping Request	Remote Address (IPv4)	192.168.1.1	PING
Network Time	SNTP Server (IPv4)	133.243.238.243	SNTP
Resolver	Host Name	japan.renesas.co	DNS
	DNS Server (IPv4)	8.8.4.4	

Powered by eForce Co.,Ltd.

Fig. 7.3 The result of sample application (HTTP server)

7.4.2 Control by the MAC controller

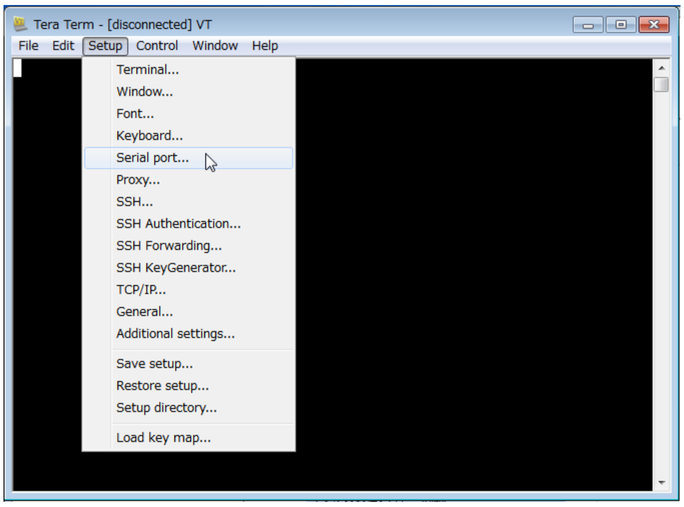
In this example, we will use the sample application "uNet3_mac". Follow the procedure below.

1. Compile and download the program and run the application.
2. Enter the commands to operate the MAC controller from the serial console.
   The command input is connected through the USB serial port and starts the terminal software.
3. Check the operation by command execution.

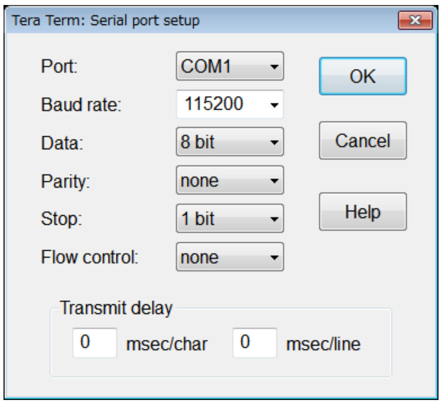
By implementing the project through the above procedure, you will be able to check the host IP address, change the mode of the PHY layer, and so on.

Setting up the terminal software

- Click the Setup tab and select Serial port



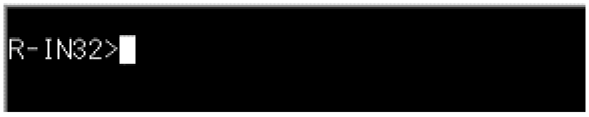
- Set up the serial port as shown below (the port setting depends on the PC you are using)


Tera Term: Serial port setup

Port: COM1
Baud rate: 115200
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none

Transmit delay
0 msec/char 0 msec/line

- If the connection is successful, the following prompt will appear.


```
R-IN32>
```

Description of available commands

ip:Displays the IP and MAC addresses of the host.

phy:Changes the mode of the PHY layer (speed, duplex, and auto-negotiation).

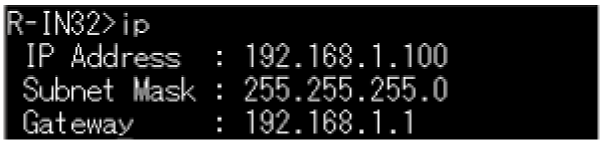
view: Displays the mode of the PHY layer (speed, duplex, link state, and auto-negotiation).

mac:Dynamically sets the reception filtering of MAC addresses.

rx:Selects or deselects dump output of Ethernet frames that are received.

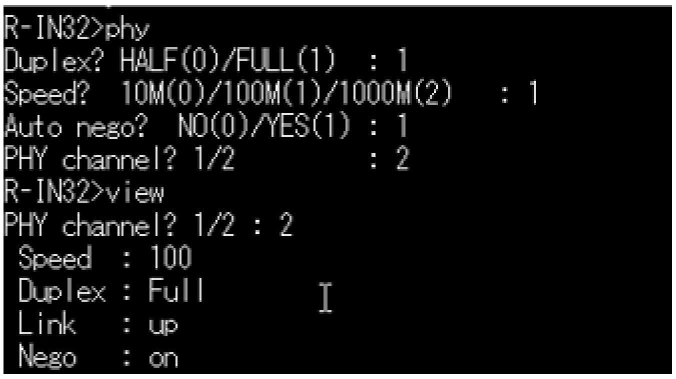
tx:Sends Ethernet frames as desired.

?:Displays the list of commands.



```
R-IN32>ip
 IP Address  : 192.168.1.100
 Subnet Mask : 255.255.255.0
 Gateway     : 192.168.1.1
```

Fig. 7.4 Example Result of Execution by the Controller (ip)



```
R-IN32>phy
Duplex? HALF(0)/FULL(1)  : 1
Speed?  10M(0)/100M(1)/1000M(2)   : 1
Auto nego?  NO(0)/YES(1) : 1
PHY channel? 1/2         : 2
R-IN32>view
PHY channel? 1/2 : 2
 Speed  : 100
 Duplex : Full
 Link   : up
 Nego   : on
```

Fig. 7.5 Example Result of Execution by the Controller (phy,view)

•Duplex? HALF(0)/FULL(1) → Change the type of communications.

FULL (full-duplex): The type of communications where data are transferred in both directions at the same time.

HALF (half-duplex): The type of communications where data are only transferred in one direction at a time so that data flow back and forth between two devices.

•Speed? → Change the speed of communications (unit: Mbps).

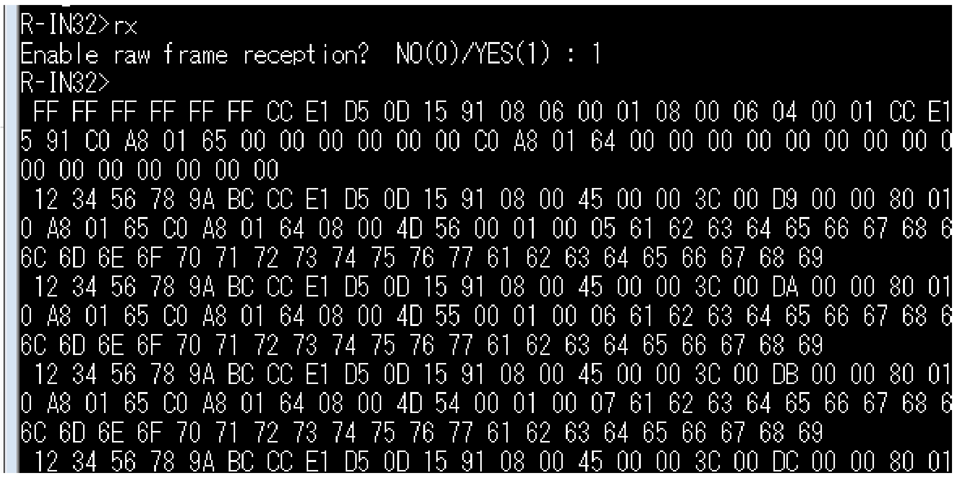
•Auto nego? NO(0)/YES(1) → Switch to auto-negotiation.

This function allows the automatic setting of communications to proceed with the optimal settings at the time of connection.

•PHY channel? 1/2 → Switch the port for which the mode is to be changed.

PHY channel 1 = PHY port 0

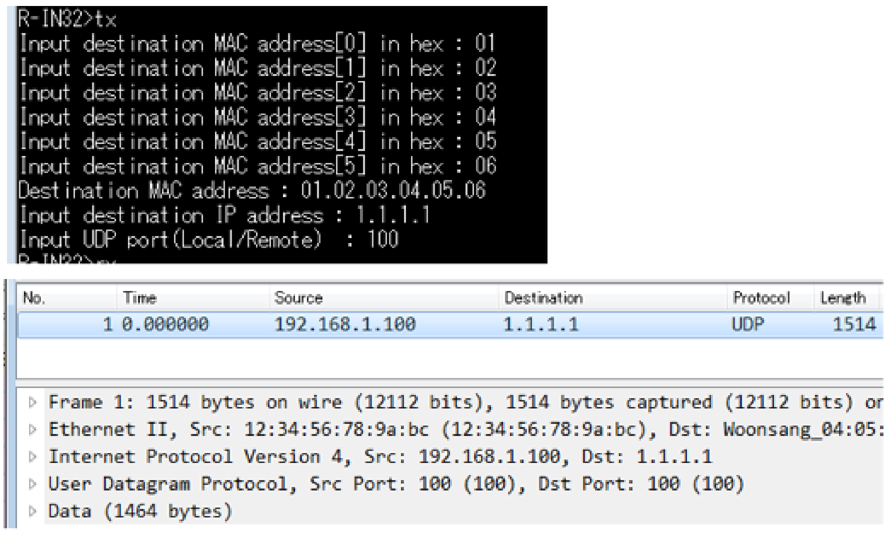
PHY channel 2 = PHY port 1



```
R-IN32>rx
Enable raw frame reception?  NO(0)/YES(1) : 1
R-IN32>
 FF FF FF FF FF FF CC E1 D5 0D 15 91 08 06 00 01 08 00 06 04 00 01 CC E1
5 91 C0 A8 01 65 00 00 00 00 00 00 C0 A8 01 64 00 00 00 00 00 00 00 00 0
00 00 00 00 00 00 00
 12 34 56 78 9A BC CC E1 D5 0D 15 91 08 00 45 00 00 3C 00 D9 00 00 80 01
0 A8 01 65 C0 A8 01 64 08 00 4D 56 00 01 00 05 61 62 63 64 65 66 67 68 6
6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69
 12 34 56 78 9A BC CC E1 D5 0D 15 91 08 00 45 00 00 3C 00 DA 00 00 80 01
0 A8 01 65 C0 A8 01 64 08 00 4D 55 00 01 00 06 61 62 63 64 65 66 67 68 6
6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69
 12 34 56 78 9A BC CC E1 D5 0D 15 91 08 00 45 00 00 3C 00 DB 00 00 80 01
0 A8 01 65 C0 A8 01 64 08 00 4D 54 00 01 00 07 61 62 63 64 65 66 67 68 6
6C 6D 6E 6F 70 71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69
 12 34 56 78 9A BC CC E1 D5 0D 15 91 08 00 45 00 00 3C 00 DC 00 00 80 01
```

Fig. 7.6 Example Result of Execution by the Controller (rx) * Example of ping execution from the R-IN32

・ Enable raw frame reception? NO(0)/YES(1) → Switch to the display of data received by the R-IN32.



```
R-IN32>tx
Input destination MAC address[0] in hex : 01
Input destination MAC address[1] in hex : 02
Input destination MAC address[2] in hex : 03
Input destination MAC address[3] in hex : 04
Input destination MAC address[4] in hex : 05
Input destination MAC address[5] in hex : 06
Destination MAC address : 01.02.03.04.05.06
Input destination IP address : 1.1.1.1
Input UDP port(Local/Remote)  : 100
```

```
No.   Time        Source          Destination   Protocol  Length
    1 0.000000    192.168.1.100   1.1.1.1       UDP       1514

▷ Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) or
▷ Ethernet II, Src: 12:34:56:78:9a:bc (12:34:56:78:9a:bc), Dst: Woonsang_04:05:
▷ Internet Protocol Version 4, Src: 192.168.1.100, Dst: 1.1.1.1
▷ User Datagram Protocol, Src Port: 100 (100), Dst Port: 100 (100)
▷ Data (1464 bytes)
```

Fig. 7.7 Example Result of Execution by the Controller (tx)
Upper half: Controller Lower half: Packet capture

Note: For data validation, use packet capture software.

•Input destination MAC address[0 to 5] in hex : → Set the destination MAC addresses.
•Input destination IP address : → Set the destination IP address.
•Input UDP port(Local/Remote) : → Set the port for data transfer.

7.4.3 BSD socket

In this example, we will use the sample application “uNet3_bsd”. Follow the procedure below.

1. Compile and download the program and run the application.
2. The command input is connected through the USB serial port and starts the terminal software.
3. Check the operation by command execution.

By implementing the project through the above procedure, you will be able to run the API of the BSD socket and check the operation through the debugging console.
Note: Checking the operation will require a separate socket on the PC (client) side to match the socket generated by the R-IN32.

The commands to be supported are as follows.

[BSD socket commands]

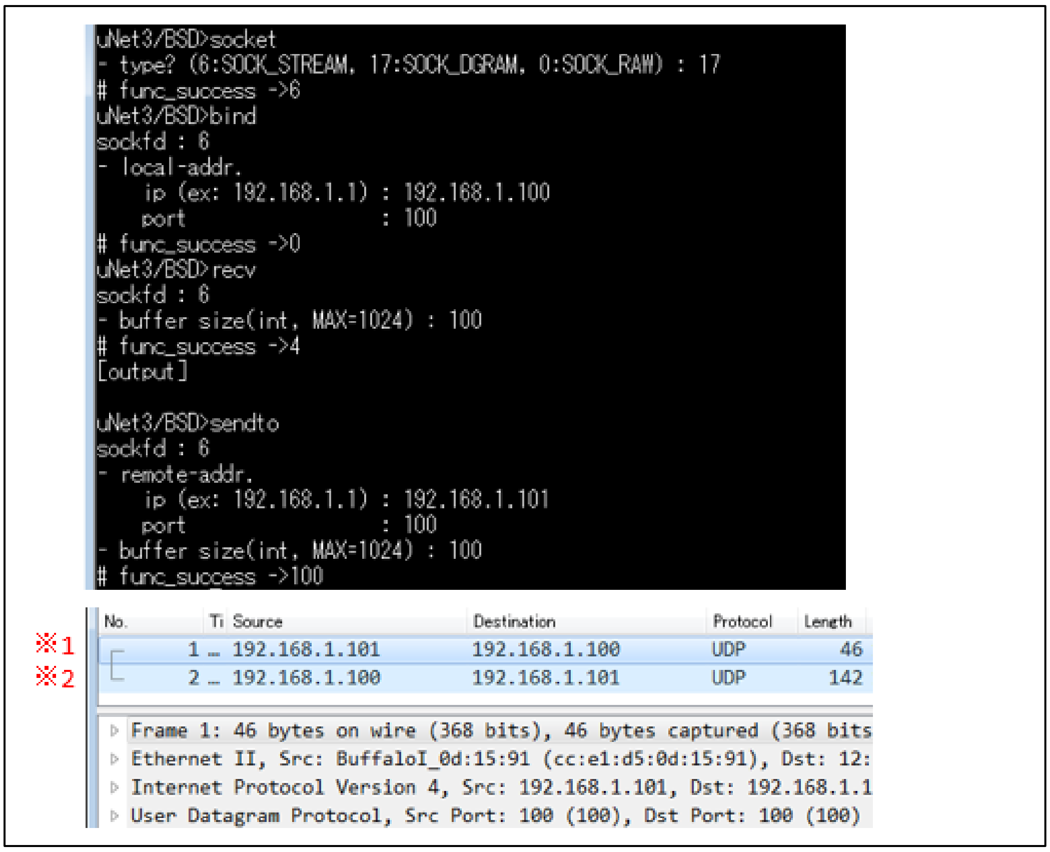
socket	Create a socket.
bind	Name a socket.
connect	Connect the socket.
listen	Listen for connections on a socket.
accept	Accept a connection to the socket.
send	Send a message to the socket.
sendto	Send a message to a socket at a specific address.
recv	Receive a message from the socket.
recvfrom	Receive a message from the socket at a specified address.
select	Monitor changes to one or more file descriptors.
shutdown	Shut down part of a full-duplex connection.
close	Close the socket.
getsockopt	Get options for the socket.
setsockopt	Set socket options.
getsockname	Get the name of the socket.
getpeername	Get the name of the peer connected to a socket.
ioctl	Control a device.

[Other commands]

netstat	Display a list of open sockets.
?	Display a list of commands.

Notes:- For the overview of the BSD socket, refer to 2.1.19, Socket.
- For how to set up the terminal software, refer to 7.4.3, Control by the MAC controller.

7.4.3.1 Transfer of packets through the UDP socket



```
uNet3/BSD>socket
- type? (6:SOCK_STREAM, 17:SOCK_DGRAM, 0:SOCK_RAW) : 17
# func_success ->6
uNet3/BSD>bind
sockfd : 6
- local-addr.
    ip (ex: 192.168.1.1) : 192.168.1.100
    port                 : 100
# func_success ->0
uNet3/BSD>recv
sockfd : 6
- buffer size(int, MAX=1024) : 100
# func_success ->4
[output]

uNet3/BSD>sendto
sockfd : 6
- remote-addr.
    ip (ex: 192.168.1.1) : 192.168.1.101
    port                 : 100
- buffer size(int, MAX=1024) : 100
# func_success ->100
```

```
     No.   Ti Source          Destination     Protocol  Length
※1     1 …  192.168.1.101   192.168.1.100   UDP           46
※2     2 …  192.168.1.100   192.168.1.101   UDP          142

▷ Frame 1: 46 bytes on wire (368 bits), 46 bytes captured (368 bits
▷ Ethernet II, Src: BuffaloI_0d:15:91 (cc:e1:d5:0d:15:91), Dst: 12:
▷ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.1
▷ User Datagram Protocol, Src Port: 100 (100), Dst Port: 100 (100)
```

Fig. 7.8 Example of Communications through UDP
Upper half: ControllerLower half: Packet capture

UDP reception

1. Create a socket.

   "socket": Create a socket.Note: Specify SOCK_DGRAM for the UDP socket.

   •type? (6:SOCK_STREAM, 17:SOCK_DGRAM, 0:SOCK_RAW) → Type of communications

   17:SOCK_DGRAM: Communications which use connection-less UDP

   Note: The return value of "# func_success ->" is the socket number of the socket that was created.

2. Set the IP address and port to bind to.

   "bind": Name a socket.

   •sockfd → Input the name of the target socket.

   •local-addr.

   ip → Set the IP address.

   port → Set the port.

3. "recv": Data reception*1

   •sockfd → Input the name of the target socket.

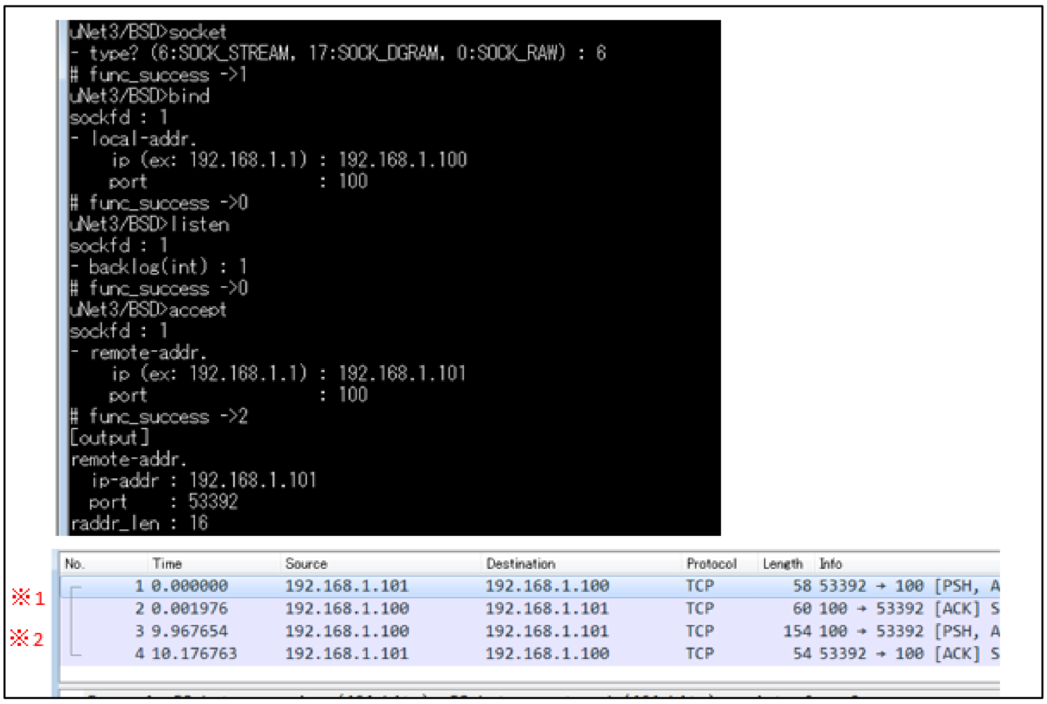
   •buffer size(int, MAX=1024) → Input the size of the reception buffer.

UDP transmission

1. Specify the destination address for the data.

   "sendto": Set the destination IP address and port, and send data corresponding to the specified size of the buffer.*2

7.4.3.2 Creating a TCP client through the TCP socket



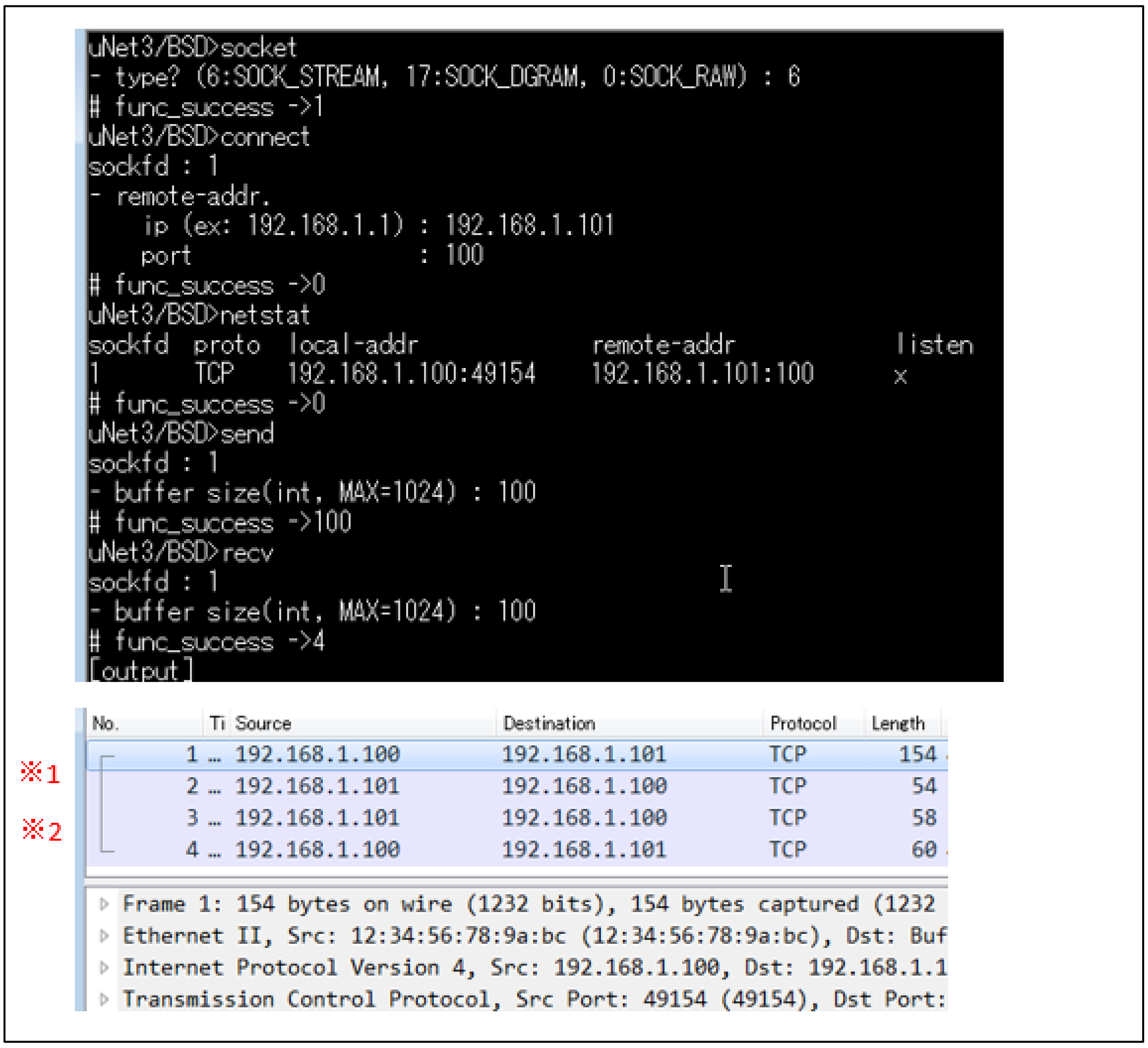
```
uNet3/BSD>socket
- type? (6:SOCK_STREAM, 17:SOCK_DGRAM, 0:SOCK_RAW) : 6
# func_success ->1
uNet3/BSD>bind
sockfd : 1
- local-addr.
    ip (ex: 192.168.1.1) : 192.168.1.100
    port                 : 100
# func_success ->0
uNet3/BSD>listen
sockfd : 1
- backlog(int) : 1
# func_success ->0
uNet3/BSD>accept
sockfd : 1
- remote-addr.
    ip (ex: 192.168.1.1) : 192.168.1.101
    port                 : 100
# func_success ->2
[output]
remote-addr.
  ip-addr : 192.168.1.101
  port    : 53392
raddr_len : 16
```

```
    No.  Time       Source          Destination     Protocol  Length Info
※1   1 0.000000   192.168.1.101   192.168.1.100   TCP           58 53392 → 100 [PSH, A
     2 0.001976   192.168.1.100   192.168.1.101   TCP           60 100 → 53392 [ACK] S
※2   3 9.967654   192.168.1.100   192.168.1.101   TCP          154 100 → 53392 [PSH, A
     4 10.176763  192.168.1.101   192.168.1.100   TCP           54 53392 → 100 [ACK] S
```

Fig. 7.9 Example of Communications through TCP (client side)
Upper half: ControllerLower half: Packet capture

1. Create a socket.
   "socket": Create a socket.Note: Specify SOCK_ STREAM for the TCP socket.
   - •type? (6:SOCK_STREAM, 17:SOCK_DGRAM, 0:SOCK_RAW) → Type of communications
     6:SOCK_STREAM: Communications which use connection TCP
     Note: The return value of "# func_success ->" is the socket number of the socket that was created.
2. Set the IP address and port to wait for connection.
   "bind": Set the name of the socket.
   - •sockfd → Input the name of the target socket.
   - •local-addr.
     Ip → Set the IP address.
     port → Set the port.
3. Wait for connection.
   "listen": Wait for connection.
   - •Sockfd → Enter the socket that waits for the connection.
     Receive an "accept" connection.
   - •Sockfd → Enter the socket that waits for the connection.
   - •Remote-addr.
     Ip → IP address setting
     port → port setting
4. Transfer data.
   "recv": Data reception*1
   "send": Data transmission*2

7.4.3.3 Creating a TCP client through the TCP socket



```
uNet3/BSD>socket
- type? (6:SOCK_STREAM, 17:SOCK_DGRAM, 0:SOCK_RAW) : 6
# func_success ->1
uNet3/BSD>connect
sockfd : 1
- remote-addr.
    ip (ex: 192.168.1.1) : 192.168.1.101
    port                 : 100
# func_success ->0
uNet3/BSD>netstat
sockfd  proto   local-addr            remote-addr          listen
1       TCP     192.168.1.100:49154   192.168.1.101:100    x
# func_success ->0
uNet3/BSD>send
sockfd : 1
- buffer size(int, MAX=1024) : 100
# func_success ->100
uNet3/BSD>recv
sockfd : 1
- buffer size(int, MAX=1024) : 100
# func_success ->4
[output]
```

```
     No.   Ti Source          Destination      Protocol  Length
        1 …  192.168.1.100   192.168.1.101    TCP       154
※1      2 …  192.168.1.101   192.168.1.100    TCP        54
※2      3 …  192.168.1.101   192.168.1.100    TCP        58
        4 …  192.168.1.100   192.168.1.101    TCP        60

▷ Frame 1: 154 bytes on wire (1232 bits), 154 bytes captured (1232
▷ Ethernet II, Src: 12:34:56:78:9a:bc (12:34:56:78:9a:bc), Dst: Buf
▷ Internet Protocol Version 4, Src: 192.168.1.100, Dst: 192.168.1.1
▷ Transmission Control Protocol, Src Port: 49154 (49154), Dst Port:
```

Fig. 7.10 Example of Communications through TCP (client side)
Upper half: ControllerLower half: Packet capture

1. Create a socket.

   "socket": Create a socket.Note: Specify SOCK_ STREAM for the TCP socket.

   •type? (6:SOCK_STREAM, 17:SOCK_DGRAM, 0:SOCK_RAW) → Type of communications

   6:SOCK_STREAM: Communications which use connection TCP

   Note: The return value of "# func_success ->" is the socket number of the socket that was created.

2. Specify the peer to which the socket is connected.

   "connect": Initiate a connection on a socket.

   •remote-addr.

   ip → Set the IP address.

   port → Set the port.

3. Transfer data.

   "recv": Data reception*1

   "send": Data transmission*2

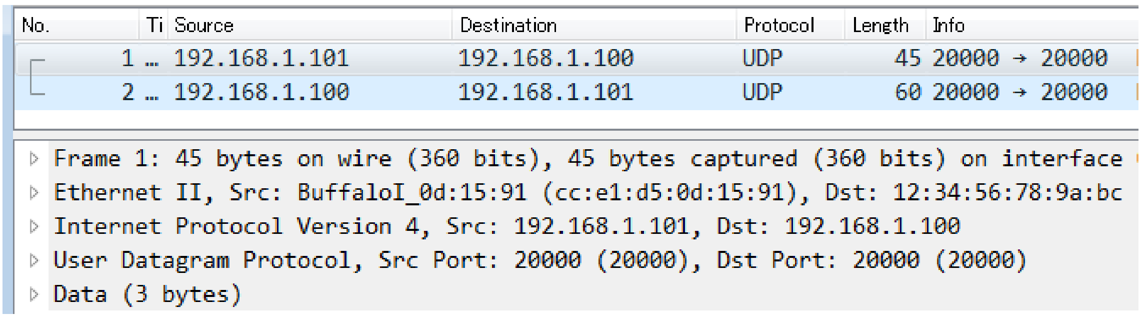
7.4.4 Non-Blocking Communications

In this example, we will use the sample application “uNet3_nonblock”. Follow the procedure below.

1. Compile and download the program and run the application.
2. Send data to the socket that was created and check the operation.

This sample application is a simple program for checking the operation of the TCP/IP stack.
It runs an echo server for the TCP and UDP sockets as a single task by using a non-blocking API for the socket.
After the program starts, the TCP socket listens on the 10000th port, while the UDP socket receives through the 20000th port. If either of these sockets receives data, the received data are returned to the source of transmission.

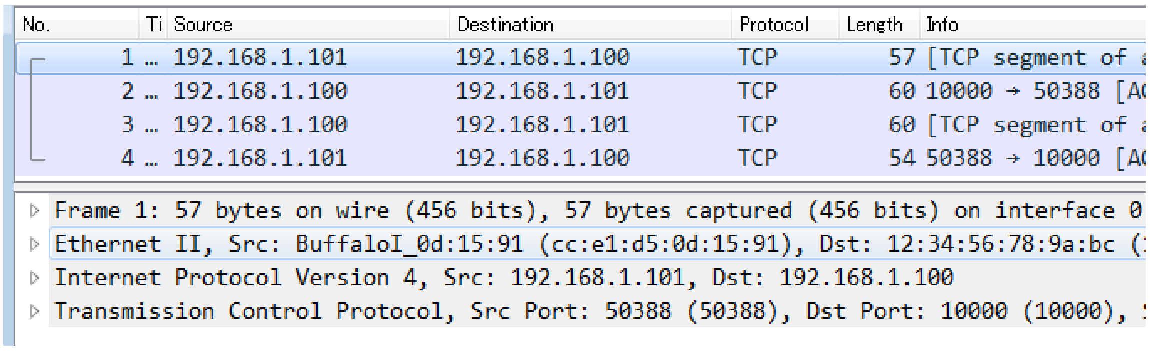
Note: - For an overview of non-blocking communications, refer to “2.1.20 Blocking and non-blocking”.
- A separate socket on the PC (client) side will be required to match the socket generated by the R-IN32.



```
No.     Ti Source          Destination      Protocol  Length Info
  1 …  192.168.1.101   192.168.1.100    UDP          45 20000 → 20000
  2 …  192.168.1.100   192.168.1.101    UDP          60 20000 → 20000

▷ Frame 1: 45 bytes on wire (360 bits), 45 bytes captured (360 bits) on interface
▷ Ethernet II, Src: BuffaloI_0d:15:91 (cc:e1:d5:0d:15:91), Dst: 12:34:56:78:9a:bc
▷ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.100
▷ User Datagram Protocol, Src Port: 20000 (20000), Dst Port: 20000 (20000)
▷ Data (3 bytes)
```

Fig. 7.11 Example of Communications through UDP



```
No.     Ti Source          Destination      Protocol  Length Info
  1 …  192.168.1.101   192.168.1.100    TCP          57 [TCP segment of
  2 …  192.168.1.100   192.168.1.101    TCP          60 10000 → 50388 [A
  3 …  192.168.1.100   192.168.1.101    TCP          60 [TCP segment of
  4 …  192.168.1.101   192.168.1.100    TCP          54 50388 → 10000 [A

▷ Frame 1: 57 bytes on wire (456 bits), 57 bytes captured (456 bits) on interface 0
▷ Ethernet II, Src: BuffaloI_0d:15:91 (cc:e1:d5:0d:15:91), Dst: 12:34:56:78:9a:bc (
▷ Internet Protocol Version 4, Src: 192.168.1.101, Dst: 192.168.1.100
▷ Transmission Control Protocol, Src Port: 50388 (50388), Dst Port: 10000 (10000),
```

Fig. 7.12 Example of Communications through TCP

8. Appendix

8.1 Packet format

(1) T_NODE

Information of communication endpoint

```
typedef struct t_node {
        UH      port;           /* Port number of socket */
        UB      ver;            /* IP version (Necessarily specify IP_VER4) */
        UB      num;            /* Device number*/
        UW      ipa;            /* IP address */
} T_NODE;
```

(2) T_NET_ADR

Information of network address

```
typedef struct t_net_adr {
        UB      ver;            /* IP- Version (Necessarily specify IP_VER4) */
        UB      mode;           /* Reserve*/
        UW      ipaddr;         /* IP Address*/
        UW      gateway;        /* Gateway*/
        UW      mask;           /* Subnet mask*/
} T_NET_ADR;
```

(3) T_NET_DEV

The information of the network device

```
typedef struct t_net_dev {
        UB          name[8];        /* Device name */
        UH          num;            /* Device number */
        UH          type;           /* Device type */
        UH          sts;            /*Reserve */
        UH          flg;            /* Reserve */
        FP          ini;            /* Pointer to dev_ini function*/
        FP          cls;            /*Pointer to dev_cls function*/
        FP          ctl;            /* Pointer to dev_ctl function*/
        FP          ref;            /*Pointer to dev_ref function*/
        FP          out;            /*Pointer to dev_snd function*/
        FP          cbk;            /*Pointer to dev_cbk function*/
        UW          *tag;           /*Reserve */
        union {                     /* MAC address */
         struct {
        UB          mac[6];
        }eth;
        } cfg;
        UH          hhdrsz;         /* Device header size */
        UH          hhdrofs;        /* Position of writing network buffer*/
        VP          opt;            / * Driver extension area */
} T_NET_DEV;
```

(4) T_NET_BUF

Information of network buffer

```
typedef struct t_net_buf {
        UW              *next;          /* Reserve */
        ID              mpfid;          /* Memory pool ID */
        T_NET           *net;           /* Network interface */
        T_NET_DEV       *dev;           /* Network device */
        T_NET_SOC       *soc;           /* Socket*/
        ER              ercd;           /* Error code */
        UH              flg;            /* Protocol stack control flag*/
        UH              seq;            /* Fragment sequence */
        UH              dat_len;        /* Data size of packet */
        UH              hdr_len;        /* Header size of packet*/
        UB              *dat;           /* Indicate data position of packet (buf) */
        UB              *hdr;           /* Indicate header position of packet (buf) */
        UB              buf[];          /* Actual packet */
} T_NET_BUF ;
```

(5) T_HOST_ADDR

Information of host address

```
typedef struct t_host_addr {
        UW      ipaddr;         /* IP address*/
        UW      subnet;         /* Subnet mask */
        UW      gateway;        /* Gateway */
        UW      dhcp;           /* DHCPserver address */
        UW      dns[2];         /* DNS address */
        UW      lease;          /* Lease period of DHCP address */
        UW      t1;             /* Renewal period of DHCP address*/
        UW      t2;             /* Rebind period of DHCP address */
        UB      mac[6];         /* MAC address */
        UH      dev_num;        /* Device number */
        UB      state;          / * DHCP client status * /
        SID     socid;          /* UDP socket ID */
} T_HOST_ADDR ;
```

(6) T_FTP_SERVER

FTP server control information

```
typedef struct t_ftp_server {

        UW      sec;            /* Security policy */


        UH      dev_num;        /* Device number */
        SID     ctl_sid;        /* Socket ID for control */
        SID     dat_sid;        /* Socket ID for data */


        ER (*auth_cbk)(UH, const char*, const char*);
        VB*     syst_name;      / * SYST command response string */
} T_FTP_SERVER ;
```

(7) T_HTTP_FILE

HTTP Content Information

```
typedef struct t_http_file {
        const char      *path;                  /* URL */
        const char      *ctype;                 /* Content type*/
        const char      *file;                  /* Content */
        Int             len;                    /* Content size*/
                                                /* HTTP callback function
        void(*cbk)(T_HTTP_SERVER *http);        or
                                                CGI handler */
        UB              ext;                    / * Extended operation flag */
} T_HTTP_FILE ;
```

(8) T_HTTP_SERVER

HTTP Server control information

```
typedef struct t_http_server {
        UW                  sbufsz;            /* Transmission buffer size */
        UW                  rbufsz;            /* Reception buffer size */
        UW                  txlen;             /* Internal data*/
        UW                  rxlen;             /* Internal data*/
        UW                  rdlen;             /* Internal data*/
        UW                  len;               /* Internal data*/
        UB                  *rbuf;             /* Transmission buffer*/
        UB                  *sbuf;             /* Reception buffer */
        UB                  *req;              /* Internal data*/
        UH                  Port;              /* Listerning port number*/
        SID                 SocketID;          /* Socket ID */
        T_HTTP_HEADER       hdr;               /* HTTP client request */
        UB                  NetChannel;        /* Device number */
        UB                  ver                /* IP version */
        UB                  server_tsk_stat    /* HTTP server task startup status */
        ID                  server_tsk_id      /* HTTP server task ID */
        struct t_http_server *next             /* HTTP server object */
        UH                  kpa_max            /* HTTP KeepAlive max value (for control) */
} T_HTTP_SERVER;
```

(9) T_HTTP_HEADER

HTTP Header information

```
typedef struct t_http_header {
        char          *method;          /* Method */
        char          *url;             /* Path name */
        char          *url_q;           /* URL query */
        char          *ver;             /* version */
        char          *host;            /* hostname */
        char          *ctype;           /* Content type */
        char          *Content;         /* Content */
        char          ContentLen;       /* Content length */
        char          kpa;              /* HTTP Keep Alive flag (for control) */
        char          *auth;            /* Authorization header */
        char          *cookie;          /* Cookie header */
} T_HTTP_HEADER;
```

(10) T_RCV_PKT_INF

Reception packet information

```
typedef struct t_rcv_pkt_inf{
        UW      src_ipa;        /* Source IP address of packet */
        UW      dst_ipa;        /* Destination IP address of packet */
        UH      src_port;       /* Source port number of packet */
        UH      dst_port;       /* Destination port number of packet */
        UB      ttl;            /* IP header TTL of packet */
        UB      tos;            /* IP header TOS of packet */
        UB      ver;            /* IP header version of packet */
        UB      num;            /* Reception device number of packet */
} T_RCV_PKT_INF;
```

(11) T_DNS_CLIENT

DNS Client information

```
typedef struct t_dns_client {
        UW      ipa;            /* DNS server IP address */
        char    *name;          /* Host name (for setting / reference) */
        UW      *ipaddr;        /* IP address (for setting / reference) */
        SID     sid;            /* DNS Socket ID (UDP) */
        UH      code:           /* Request RR type */
        UB      dev_num;        /* Device number */
        UB      retry_cnt;      /* number of retries */
} T_DNS_CLIENT ;
```

(12) T_DHCP_CLIENT

DHCP Client information

```
typedef struct t_dhcp_client {
        T_DHCP_CTL      ctl             /* Internal data */
        UW              ipaddr;         /* IP address */
        UW              subnet;         /* Subnet mask */
        UW              gateway;        /* Gateway */
        UW              dhcp;           /* DHCPserver address */
        UW              dns[2];         /* DNS address */
        UW              lease;          /* Lease period of DHCP address */
        UW              t1;             /* Renewal period of DHCP address */
        UW              t2;             /* Rebind period of DHCP address */
        UB              mac[6];         /* MAC address */
        UH              dev_num;        /* Device number */
        UB              state;          /* DHCP client status */
        SID             socid;          /* UDP socket ID */
        UB              arpchk;         /* APR check */
        T_DHCP_UOPT     *uopt;          /* DHCP option acquisition parameters */
        UB              uopt_len;       /* Number of DHCP option acquisition parameters */
        UB              retry_cnt;      /* number of retries */
} T_DHCP_CLIENT;
```

(13) T_DHCP_UOPT

DHCP acquisition option information

```
typedef struct t_dhcp_uopt {
        T_DHCP_CTL      ctl             /* Internal data */
        UB              code;           /* DHCP option code */
        UB              len;            /* Optional element size / piece */
        UB              ary;            /* Number of optional elements */
        UB              flag;           /* Element discrimination / status flag */
        VP              val;            /* Element storage destination pointer */
} T_DHCP_UOPT;
```

(14) T_PING_CLIENT

Ping Client Information

```
typedef struct  t_ping_client {
        SID     sid;            /* ICMP Socket ID */
        UW      ipa;            /* Destination IP Address */
        TMO     tmo;            /* Response Time out (ms) */
        UH      devnum;         /* Device number */
        UH      len;            /* Packet Size (byte) */
} T_PING_CLIENT;
```

(15) T_SNTP_CLIENT

SNTP Client Information

```
typedef struct t_sntp_client {
        SID             sid;            /* Socket ID */
        UW              ipa;            /* SNTP server IP address */
        TMO             tmo;            /* Response Time out */
        UH              devnum;         /* Device numbr */
        UH              port;           /* Port number */
        UB              ipv;            /* IP version */
        UB              stt;            /* Received data Stratum field value */
} T_SNTP_CLIENT;
```

8.2 Constant and Macro

(1) IP Address

ADDR_ANY	IP address 0
IP_VER4	IP version 4

(2) Port Number

PORT_ANY	Port number 0

(3) IP protocol

IP_PROTO_TCP	TCP protocol
IP_PROTO_UDP	UDP protocol
IP_PROTO_ICMP	ICMP protocol

(4) Network interface control

NET_IP4_CFG	Configure and verify IP Address, Subnet mask
NET_IP4_TTL	Configure and vefiry TTL
NET_BCAST_RCV	Configure and verify reception of broadcast
NET_MCAST_JOIN	Join in multicast group
NET_MCAST_DROP	Drop from multicast Group
NET_MCAST_TTL	Configure TTL used in multicast transmission

(5) Parameter of socket

SOC_IP_TTL	Configure and verify TTL of Socket
SOC_IP_TOS	Configure and verify TOS of Socket
SOC_TMO_SND	Configure and verify blocking time-out of snd_soc
SOC_TMO_RCV	Configure and verify blocking time-out of rcv_soc
SOC_TMO_CON	Configure and verify blocking time-out of con_soc
SOC_TMO_CLS	Configure and verify blocking time-out of cls_soc
SOC_IP_LOCAL	Get port number and IP address of local host
SOC_IP_REMOTE	Get port number and IP address of remote host
SOC_CBK_HND	Register callback function
SOC_CBK_FLG	Specify callback event
SOC_RCV_PKT_INF	Get information of reception packet

(6) Connection mode of socket

SOC_CLI	Connect to remote host (active connection)
SOC_SER	Wait for connection (passive connection)

(7) Termination mode of socket

SOC_TCP_CLS	Disconnect socket. (Terminate connection)
SOC_TCP_SHT	Disable only the transmission process. Reception is possible

(8) Interruption mode of socket

SOC_ABT_CON	Abort con_soc()
SOC_ABT_CLS	Abort cls_soc()
SOC_ABT_SND	Abort snd_soc()
SOC_ABT_RCV	Abort rcv_soc()
SOC_ABT_ALL	Abort all the processes of socket

(9) Callback Event

EV_SOC_CON	Enable con_soc() to be non-blocking mode
EV_SOC_CLS	Enable cls_soc() to be non-blocking mode
EV_SOC_SND	Enable snd_soc() to be non-blocking mode
EV_SOC_RCV	Enable rcv_soc() to be non-blocking mode

8.3 Error Code List

E_NOSPT	-9	Unsupported function
E_PAR	-17	Parameter error
E_ID	-18	Illegal ID number
E_NOMEM	-33	Insufficient memory
E_OBJ	-41	Object status error
E_NOEXS	-42	Uncreated object
E_QOVR	-43	Queuing overflow
E_RLWAI	-49	Forced cancellation of wait state
E_TMOUT	-50	Polling failure or time-out
E_CLS	-52	Change status of waiting object
E_WBLK	-57	Non-blocking acceptance
E_BOVR	-58	Buffer overflow
EV_ADDR	-98	Default G/W not set

8.4 API List

API Name	
A) Network Interface	
net_ini	Initialize TCP / IP protocol stack
net_cfg	Configure parameters of network interface
net_ref	Refer parameters of network interface
net_acd	Detection IP Address Confliction
B) Network Device Control	
net_dev_ini	Initialize network device
net_dev_cls	Release Network Device
net_dev_ctl	Control network device
net_dev_sts	Get status of network device
C) Socket	
cre_soc	Create socket (Standard version only)
del_soc	Delete a socket (Standard version only)
con_soc	Socket connection
cls_soc	Socket interruption
snd_soc	Send data
rcv_soc	Receive data
cfg_soc	Configure parameter of socket
ref_soc	Refer parameter of socket
abt_soc	Abort process of socket
soc_ext	stop process of socket all at once
D) Network Application	
dhcp_client	Start DHCP Client
ftp_server	Start FTP Server
ftp_server_stop	Stop FTP server
http_server	Start HTTP server
http_server_stop	Stop HTTP server
CgiGetParam	Analyze CGI argument
CgiGetParamN	Analyze CGI argument
CookieGetItem	Parsing cookie headers
HttpSendText	Send text content
HttpSendFile	Send Attached File
HttpSendResponse	Send Image Content
HttpSetContent	Addition of transmission buffer for HTTP server control information
HttpSetContentKpa	Add HTTP Keep-Alive header to send buffer
HttpSetContentCookie	Add Set-Cookie header to send buffer
HttpSendBuffer	Send specified buffer contents
dns_get_ipaddr	Get IP address from host name
dns_get_name	Get host name from IP address
dns_query_ext	Issuing DNS queries
dhcp_bind	Get DHCP Lease Information
dhcp_renew	Renewal DHCP lease information
dhcp_rebind	Application for reassignment of lease information
dhcp_reboot	Reboot DHCP client
dhcp_release	Release DHCP lease information

dhcp_inform	Get DHCP option
ping_client	ICMP Echo request and response
sntp_client	Get NTP time
E) Others	
htons	Convert 16-bit value to network byte order
ntohs	Convert 16-bit value to host byte order
htonl	Convert 32-bit value to network byte order
ntohl	Convert 32- bit value to host byte order
ip_aton	Convert IPv4 address string in dot notation to 32-bit value
ip_ntoa	Convert 32-bit value IPv4 address to IPv4 address string in dot notation
ip_byte2n	Convert IPv4 address array to 32 bit value
ip_n2byte	Convert 32-bit value IPv4 addresse to array
arp_set	Setting static ARP entries
arp_ref	ARP cache reference
arp_req	Send ARP request
arp_clr	Clear ARP cache
arp_del	Delete ARP entry
net_atoi	Numerical conversion of character string to int type
net_atol	Numerical conversion of character string to long type
net_itoa	String conversion of int type number
net_strncasecmp	Comparison of character strings (identification of uppercase and lowercase letters)
net_strcmp	String comparison
net_strncmp	String comparison
net_strcpy	Copy of string
net_strlen	Get string length
net_strncat	String concatenation
net_strcat	String concatenation
net_strchr	Character search
net_strstr	Search for strings
net_strcasestr	Character string search Uppercase and lowercase letters are identified
net_strncpy	N-character copy of a string

8.5 Resource list

8.5.1 Kernel objects

(1) Kernel object used by Ethernet device driver

Object	Object ID	Description
Task	ID_TASK_ETH_SND	Ether driver send task (stack size: 1024Byte)
Task	ID_TASK_ETH_RCV	Ether driver receive task (stack size: 1024Byte)
Task	ID_TASK_PHY0_LINK	PHY driver control task (stack size: 512Byte)
Task	ID_TASK_PHY1_LINK	PHY driver control task (stack size: 512Byte)
Event flag	ID_FLG_ETH_RX_MAC	Ether driver event flag
Event flag	ID_FLG_ETH_TX_MAC	Ether driver event flag
Event flag	ID_FLG_PHY_STS	Ether driver event flag
Event flag	ID_FLG_SYSTEM	Ether driver event flag
Mail box	ID_MBX_ETH_SND	Ether driver mail box
Mail box	ID_MBX_ETH_MEMPOL	Ether driver mail box

(2) Kernel object used by TCP/IP protocol stack (μ Net3 compatible)

Object	Object ID	Description
Task	ID_TASK_TCP_TIM	TCP/IP stack time management task for R-IN32
Semaphore	ID_SEM_TCP	Semaphore to control protocol stack resource.

(3) Kernel object used by TCP/IP protocol stack (BSD compatible)

Object	Object ID	Description
Task	ID TSK_BSD_API	BSD Wrapper task
Task	ID_LO_IF_TSK	Loop back device task
Mail box	ID MBX_BSD_REQ	BSD Wrapper communication between task
Mail box	ID_LO_IF_MBX	Communication between device task

Note Kernel objects used by TCP/IP protocol stack (uNet3 compatible) are also used added to above.

(4) Kernel object used by memory management

Object	Object ID	Description
Mail box	ID_MBX_ETH_MEMPOL	Memory management

8.5.2 Hardware ISR

Table.8.1 Hardware ISR used by TCP/IP stack

Hardware ISR reason	Operation	Description
PHY0_IRQn	set_flg()	PHY driver
PHY1_IRQn	set_flg()	PHY driver
ETHTXDMA_IRQn	set_flg()	Ethernet driver send operation
ETHTXDERR_IRQn	set_flg()	Ethernet driver send operation
ETHTX_IRQn	set_flg()	Ethernet driver send operation
ETHTXFIFO_IRQn	set_flg()	Ethernet driver send operation
ETHTXFIFOERR_IRQn	set_flg()	Ethernet driver send operation
ETHRXDMA_IRQn	set_flg()	Ethernet driver receive operation
ETHRXFIFO_IRQn,	set_flg()	Ethernet driver receive operation
ETHRXDERR_IRQn	set_flg()	Ethernet driver receive operation
ETHRXERR_IRQn	set_flg()	Ethernet driver receive operation

REVISION HISTORY	R-IN32M4-CL3 Series User’s Manual: TCP/IP stack

Rev.	Date	Description	
		Page	Summary
1.00	Aug 31, 2021	-	First edition issued (For R-IN32M4-CL3 only)

[Memo]

R-IN32M4-CL3 Series User’s Manual: TCP/IP stack

Publication Date: Rev.1.00 Aug. 31, 2021

Published by: Renesas Electronics Corporation

R-IN32M4-CL3 Series
User's Manual: TCP/IP stack

Renesas Electronics Corporation