Modem Gen93 NAL Programming Guide

Version: 0.3

Release date: 2017-07-10

Specifications are subject to change without notice.

MediaTek Confidential A

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Description |
| 0.1 | 2017-02-14 | Christine Tsai | Initial Draft. |
| 0.2 | 2017-04-15 | Christine Tsai | Update usage |
| 0.3 | 2017-06-23 | Christine Tsai | Update features |

MediaTek Confidential A

Table of Contents

[1 Introduction 6](#_Toc475394555)

[**1.1 Developing Environment 6**](#_Toc475394556)

[2 NAL APIs and Definitions 7](#_Toc475394557)

[**2.1 Brief Introduction of NAL APIs 8**](#_Toc475394558)

[3 Programming Guide for Network Abstraction Layer (NAL) Interface 17](#_Toc475394559)

[**3.1 Client Side 17**](#_Toc475394560)

[**3.1.1 UDP Client 17**](#_Toc475394561)

[**3.1.2 TCP Client 17**](#_Toc475394562)

[**3.2 Server Side 17**](#_Toc475394563)

[**3.3 NAL 5-tuple Filter 17**](#_Toc475394564)

[**3.3.1 How to use the NAL filter APIs? 17**](#_Toc475394565)

[**3.4 User Scenarios 18**](#_Toc475394566)

[**3.4.1 UDP client socket 18**](#_Toc475394567)

[**3.4.2 TCP client socket 20**](#_Toc475394568)

[**3.5 Sample Codes of NAL Socket Programming 21**](#_Toc475394569)

[**3.5.1 UDP client socket with NAL filter (part) 21**](#_Toc475394570)

[**3.5.2 TCP client socket with NAL filter (part) 23**](#_Toc475394571)

Lists of Tables and Figures

[Table 1. NAL API List 7](#_Toc475379571)

[Table 2. NAL Structure List 7](#_Toc475379572)

MediaTek Confidential A

[Figure 1. An example for UDP client socket 20](#_Toc475436666)

[Figure 2. An exaSmple for TCP client socket 21](#_Toc475436667)

# Introduction

Inside the network transport layer of Nucleus operating system, we have adopted NU socket APIs to create sockets, to establish TCP/UDP connections and deal with all other work that transport layer does. However, this can only work well for internal use.

After taking some security and business concerns into consideration, an abstraction layer inside the transport layer that covers NU socket APIs is necessary; moreover, it reduces the time customers get involved into the development, since they don’t need to be aware of what the lower layer actually looks like.

Therefore, Network Abstraction Layer (NAL) is created as the MTK solution and we are going to introduce all NAL APIs and structures in Chap2 , giving some examples about the usage of NAL APIs in Chap3.

## Developing Environment

This document is basically for Gen93 Modem users, especially for BIP developers.

# NAL APIs and Structures

Kindly access to *Table 1.* & *Table 2.* to find all NAL APIs and structures.

|  |  |
| --- | --- |
| Functions | |
| Socket Interface | |
| nal\_socket | nal\_getsockopt |
| nal\_bind | nal\_setsockopt |
| nal\_get\_sock\_name | nal\_sendmsg |
| nal\_connect | nal\_recvmsg |
| nal\_is\_connected | nal\_shutdown |
| nal\_send | nal\_close\_socket |
| nal\_recv | nal\_fcntl |
| nal\_send\_to | nal\_fd\_check |
| nal\_recv\_from | nal\_fd\_init |
| nal\_fd\_reset | nal\_select |
| nal\_fd\_set |  |
| Socket Utilities | |
| nal\_inet\_ntop | nal\_inet\_pton |
| nal\_htons | nal\_ntohs |
| nal\_htonl | nal\_ntohl |
| NAL Filter | |
| nal\_add\_filter | nal\_delete\_filter |
| nal\_update\_filter |  |

Table . NAL API List

|  |  |
| --- | --- |
| Structures | |
| nal\_fdset | nal\_iovec |
| nal\_msghdr | nal\_id\_struct |
| nal\_addr\_struct | nal\_sockaddr\_struct |
| nal\_linger\_struct | nal\_filter\_type\_t |
| nal\_filter\_info\_t |  |

Table . NAL Structure List

## Brief Introduction of NAL APIs

### NAL Socket Interface

|  |  |
| --- | --- |
| Function Name | nal\_socket |
| Synopsis | nal\_int nal\_socket (nal\_int16 family, nal\_int16 type, nal\_int16 protocol); |
| Description | Create a new NAL socket. |
| Return Value | On success, a socket descriptor (>= 0) is returned; on error, *NAL\_INVALID\_PROTOCOL* or *NAL\_NO\_SOCK\_MEMORY*  is returned. |
| Note | 1. The socket ID is registered in the global socket structure. 2. A port number is given by default in TCPIP stack. |

|  |  |
| --- | --- |
| Function Name | nal\_bind |
| Synopsis | nal\_status nal\_bind (nal\_int socketd, nal\_addr\_struct\* myaddr, nal\_int16 addrlen); |
| Description | Assign a local address to a NAL socket. |
| Return Value | On success, the socket descriptor (>= 0) is returned; on error, the corresponding error number (*NAL\_INVALID\_PARM/NAL\_INVALID\_SOCKET/NAL\_INVALID\_PORT/NAL\_INVALID\_ADDRESS*) is returned. |
| NOTE | 1. We must take this function before setting NAL filter in both UDP and TCP case. 2. If the NAL\_SO\_REUSEADDR has been set and the address is IP\_ADDR\_ANY, the port number can be repeated. |

|  |  |
| --- | --- |
| Function Name | nal\_get\_sock\_name |
| Synopsis | nal\_status nal\_get\_sock\_name (nal\_int socketd, nal\_sockaddr\_struct\* localaddr, nal\_int16\* addrlen); |
| Description | Get the current address to which the *socketd* is bound. |
| Return Value | On success, *NAL\_SUCCESS* is returned; on error, the corresponding error number (*NAL\_INVALID\_PARM/NAL\_INVALID\_SOCKET/NAL\_NOT\_CONNECTED*) is returned. |

|  |  |
| --- | --- |
| Function Name | nal\_connect |
| Synopsis | nal\_status nal\_connect (nal\_int socketd, nal\_addr\_struct\* servaddr, nal\_int16 addrlen); |
| Description | Called by a client to establish a connection to a server. |
| Return Value | On success, the socket descriptor (>=0) is returned; on error, the corresponding error number (*NAL\_INVALID\_PARM/NAL\_INVALID\_SOCKET/NAL\_NOT\_CONNECTED/NAL\_INVALID\_ADDRESS/NAL\_IS\_CONNECTING/other IMCP error codes\*1*) is returned. |
| Note | 1. nal\_connect() can only be used once by one socket, or it will return NAL\_INVALID\_SOCKET. 2. Only establish the connection under TCP protocol. 3. It only checks if the source address is exist under UDP protocol. |

|  |  |
| --- | --- |
| Function Name | nal\_is\_connected |
| Synopsis | nal\_status result = nal\_is\_connected (nal\_int socketd); |
| Description | Check if the socket is connected or not. It is used by TCP client to check if the connection has established. |
| Return Value | On success, *NAL\_TRUE* is returned; on error, *NAL\_FALSE* or *NAL\_INVALID\_SOCKET* is returned. |
| Note | 1. If NAL\_FALSE is return, it means that the socket is not connected or is still connecting to the server. 2. While using nal\_is\_connected(), the return value is likely to change as the current connection status changes; however, if one side crashes unexpectedly, the other side won’t be notified. |

|  |  |
| --- | --- |
| Function Name | nal\_send |
| Synopsis | nal\_int32 nal\_send (nal\_int socketd, nal\_char\* buff, nal\_uint16 nbytes, nal\_int16 flags); |
| Description | This system call is used to transmit a message (the MAX size is 65496 bytes) to another socket and can only be used when the socket is in a *connected* state. |
| Return Value | On success, the number of bytes sent (>0) is returned; on error, the corresponding error number (*NAL\_INVALID\_SOCKET/NAL\_NOT\_CONNECTED/NAL\_INVALID\_PARM/NAL\_WOULD\_BLOCK/NAL\_NO\_DATA\_TRANSFER/other IMCP error codes\*1*) is returned. |
| Note | 1. This function can be used under both TCP and UDP protocol. 2. Under TCP protocol, the TCPIP stack would resend data until all data are sent if NAL\_SF\_BLOCK flag is set. (Basically, the flag is set by default) 3. On success, TCP send must get the value (>0); UDP get the value (>=0). |

|  |  |
| --- | --- |
| Function Name | nal\_recv |
| Synopsis | nal\_int32 nal\_recv (nal\_int socketd, nal\_char\* buff, nal\_uint16 nbytes, nal\_int16 flags); |
| Description | This system call is used to receive a message from a socket and is used only on a connected socket. |
| Return Value | On success, number of bytes received is returned; on error, the corresponding error number (*NAL\_NO\_PORT\_NUMBER*/*NAL\_INVALID\_SOCKET/NAL\_INVALID\_OPTION/NAL\_NOT\_CONNECTED/NAL\_WOULD\_BLOCK*) is returned. |
| Note | 1. This function can be used under both TCP and UDP protocol. 2. The receive number should be (>=0) on success. It is not guarantee that the byte received is the same as the max number of bytes of data. 3. It’s not necessary to do nal\_connect() before nal\_send() under UDP protocol. |

|  |  |
| --- | --- |
| Function Name | nal\_send\_to |
| Synopsis | nal\_int32 nal\_send\_to (nal\_int socketd, nal\_char\* buff, nal\_uint16 nbytes, nal\_int16 flags, nal\_addr\_struct\* to, nal\_int16 addr\_len); |
| Description | This system call is used to transmit a message to another socket in a connectionless transfer. |
| Return Value | On success, number of bytes sent is returned; on error, the corresponding error number (*NAL\_INVALID\_SOCKET/NAL\_INVALID\_PARM/NAL\_INVALID\_ADDRESS/NAL\_NOT\_CONNECTED/NAL\_NO\_DATA\_TRANSFER/NAL\_DEVICE\_DOWN/NAL\_WOULD\_BLOCK*) is returned. |
| Note | 1. For UDP protocol only. 2. On success, the value must (>=0). |

|  |  |
| --- | --- |
| Function Name | nal\_recv\_from |
| Synopsis | nal\_int32 nal\_recv\_from (nal\_int socketd, nal\_char\* buff, nal\_uint16 nbytes, nal\_int16 flags, nal\_addr\_struct\* from, nal\_int16\* addr\_len); |
| Description | This system call is used to receive a message from a socket in a connectionless transfer. |
| Return Value | On success, this call returns the number of bytes received; on error, the corresponding error number (NAL\_INVALID\_PARM/NAL\_NOT\_CONNECTED/NAL\_NO\_DATA\_TRANSFER/NAL\_DEVICE\_DOWN/NAL\_WOULD\_BLOCK/other IMCP error codes\*1) is returned. |
| Note | 1. It’s not necessary to do nal\_connect() before nal\_send\_to() under UDP protocol. |

|  |  |
| --- | --- |
| Function Name | nal\_sendmsg |
| Synopsis | nal\_int32 nal\_sendmsg (nal\_int socketd, nal\_msghdr\* msg, nal\_int16 flags); |
| Description | This system call is used to send a message from a socket, parsing incoming ancillary data, doing scatter write via the *nal\_msghdr* structure. |
| Return Value | On success, this call returns the number of bytes send; on error, the corresponding error number (*NAL\_NO\_PORT\_NUMBER/NAL\_INVALID\_SOCKET/NAL\_INVALID\_PARM/NAL\_NOT\_CONNECTED/NAL\_INVALID\_ADDRESS/NAL\_NO\_DATA\_TRANSFER/NAL\_DEVICE\_DOWN/NAL\_WOULD\_BLOCK/other IMCP error codes\*1*) is returned. |
| Note | 1. Please refer to the not in nal\_send(). 2. It can be used under IPRAW protocol as well. |

|  |  |
| --- | --- |
| Function Name | nal\_recvmsg |
| Synopsis | nal\_int32 nal\_recvmsg (nal\_int socketd, nal\_msghdr\* msg, nal\_int16 flags); |
| Description | This system call is used to receive data from a socket, creating and returning an ancillary data structure for a non-TCP socket. In addition, it can do scatter read using the *msghdr* structure. |
| Return Value | On success, this call returns the number of bytes received; on error, the corresponding error number (*NAL\_NO\_PORT\_NUMBER/NAL\_INVALID\_SOCKET/NAL\_INVALID\_PARM/NAL\_INVALID\_OPTION/NAL\_NOT\_CONNECTED/NAL\_NO\_DATA\_TRANSFER/NAL\_DEVICE\_DOWN/NAL\_WOULD\_BLOCK/NAL\_NO\_ROUTE\_TO\_HOST/NAL\_CONNECTION\_REFUSED/NAL\_MSG\_TOO\_LONG/NAL\_CONNECTION/TIMED\_OUT/other IMCP error codes\*1*) is returned. |
| Note | 1. Please refer to the not in nal\_recv(). It can be used under IPRAW protocol as well. |

|  |  |
| --- | --- |
| Function Name | nal\_getsockopt |
| Synopsis | nal\_status nal\_getsockopt (nal\_int socketd, nal\_int level, nal\_int optname, void\* optval, nal\_int optlen); |
| Description | Get the current setting of socket flags by the socket descriptor *socketd*. |
| Return Value | On success, *NAL\_SUCCESS* is returned; on error, the corresponding error number (*NAL\_INVALID\_PARM/NAL\_INVALID\_OPTION/NAL\_INVAL/NAL\_INVALID\_LEVEL*) is returned. |

|  |  |
| --- | --- |
| Function Name | nal\_setsockopt |
| Synopsis | nal\_status nal\_setsockopt (nal\_int socketd, nal\_int level, nal\_int optname, void\* optval, nal\_int optlen); |
| Description | Set options for the socket referred to by the socket descriptor *socketd*. |
| Return Value | On success, NAL\_SUCCESS is returned; on error, the corresponding error number (NAL\_INVALID\_SOCKET/NAL\_INVALID\_OPTION/NAL\_INVAL/NAL\_INVALID\_LEVEL/NAL\_MEM\_ALLOC/NAL\_ADDRINUSE/NAL\_UNVAILABLE) is returned. |

|  |  |
| --- | --- |
| Function Name | nal\_shutdown |
| Synopsis | nal\_status nal\_shutdown (nal\_int socketd, nal\_int how); |
| Description | This system call causes all or part of a full-duplex connection on the socket associated with *soctetd* to be shut down. |
| Return Value | On success, *NAL\_SUCCESS* is returned; on error, *NAL\_INVALID\_SOCKET* or *NAL\_NOT\_CONNECTED* is returned. |
| Note | 1. You can fill in NAL\_SHUT\_RD/NAL\_SHUT\_WR/NAL\_SHUT\_RDWR as the second parameter. |

|  |  |
| --- | --- |
| Function Name | nal\_close\_socket |
| Synopsis | nal\_status nal\_close\_socket (nal\_int socketd); |
| Description | Breaking the socket connections between NAL sockets by *socketd*. |
| Return Value | On success, *NAL\_SUCCESS* is returned; on error, *NAL\_INVALID\_SOCKET* or *NAL\_NOT\_CONNECTED* is returned. |
| Note | 1. Basically, the nal\_close\_socket() closes the socket in non-blocking way. 2. If you want to make sure the TCP connection of the specific socket has been closed correctly, set NAL\_SO\_LINGER before closing the socket, or it may cause problem while creating new sockets right after nal\_close\_socket(). |

|  |  |
| --- | --- |
| Function Name | nal\_fcntl |
| Synopsis | nal\_status nal\_fcntl (nal\_int socketd, nal\_int16 command, nal\_int16 argument); |
| Description | This function is used to enable/disable the NAL socket operational flags for *socketd*. |
| Return Value | On success, *NAL\_SUCCESS* is returned; on error, NAL\_INVALID\_SOCKET or NAL\_NO\_ACTION is returned. |
| Note | 1. List flag bits supported now:    1. *NAL\_SF\_BLOCK*    2. *NAL\_SF\_LISTENER*    3. *NAL\_SF\_ZC\_MODE*    4. *NAL\_SF\_V4\_MAPPED*    5. *NAL\_SF\_BIND* |

|  |  |
| --- | --- |
| Function Name | nal\_fd\_check |
| Synopsis | nal\_int nal\_fd\_check (nal\_int socketd, nal\_fdset\* fd); |
| Description | This function is used to check if a particular bit has been set in a bit map. |
| Return Value | If the bit has been set, it returns NAL\_TRUE; otherwise, it returns NAL\_FALSE. |

|  |  |
| --- | --- |
| Function Name | nal\_fd\_init |
| Synopsis | void nal\_fd\_init (nal\_fdset\* fd); |
| Description | Set all bits inside a bit map to zero. |
| Return Value | N/A |

|  |  |
| --- | --- |
| Function Name | nal\_fd\_reset |
| Synopsis | void nal\_fd\_reset (nal\_int socketd, nal\_fdset\* fd); |
| Description | Reset the specified bit which the socket points to. |
| Return Value | N/A |

|  |  |
| --- | --- |
| Function Name | nal\_fd\_set |
| Synopsis | void nal\_fd\_set (nal\_int socketd, nal\_fdset\* fd); |
| Description | Set the specified bit which the socket points to. |
| Return Value | N/A |

|  |  |
| --- | --- |
| Function Name | nal\_select |
| Synopsis | nal\_status nal\_select (nal\_int max\_sockets, nal\_fdset\* readfs, nal\_fdset\* writefs, nal\_fdset\* exceptfs, nal\_unsigned timeout); |
| Description | This function is used to check for data on multiple sockets. |
| Return Value | On success, *NAL\_SUCCESS* is returned; on error, the corresponding error number is returned. |
| Note | 1. If the function returns NAL\_SUCCESS, it means that at least one socket is ready for data. 2. If there is not data suspending to R/W and no socket error is pending, the FD bit would be reset. Therefore, make sure to set the FD bit again before another nal\_select(). |

|  |  |
| --- | --- |
| Function Name | nal\_add\_filter |
| Synopsis | nal\_status nal\_add\_filter(nal\_int socketd, nal\_addr\_struct\* dest); |
| Description | This function is used to register a filter to IPcore for the specific socket. If “*dest”* is “NULL”, nal\_add\_filter will pass the 3-tuple information to IPcore. |
| Return Value | On success, the filter ID (>=0) is returned; on error, the corresponding NAL Filter error code\*2 is returned. |

|  |  |
| --- | --- |
| Function Name | nal\_delete\_filter |
| Synopsis | nal\_status nal\_delete\_filter(nal\_int socketd, nal\_int filter\_id); |
| Description | This function is used to deregister a filter from IPcore. If set “NAL\_DEL\_ALL” to the second argument, it will delete all NAL filters the *socketd* indicates to. |
| Return Value | On success, NAL\_SUCCESS is returned; on error, the corresponding NAL Filter error code\*2 is returned. |

|  |  |
| --- | --- |
| Function Name | nal\_update\_filter |
| Synopsis | nal\_status nal\_update\_filter(nal\_int socketd, nal\_int filter\_id, nal\_addr\_struct\* dest); |
| Description | Delete the NAL filter which *filter\_id* dedicated to, and set the new NAL filter to *dest*. |
| Return Value | On success, the filter ID (>=0) is returned; on error, the corresponding NAL Filter error code\*2 is returned. |

### NAL Socket Utilities

|  |  |
| --- | --- |
| Function Name | nal\_inet\_ntop |
| Synopsis | nal\_status nal\_inet\_ntop (nal\_int family, void\* src, nal\_char\* dst, nal\_int size); |
| Description | Format the IP address into an ASCII string. |
| Return Value | On success, *NAL\_SUCCESS* is returned; on error, *NAL\_INVALID\_PARM* is returned. |

|  |  |
| --- | --- |
| Function Name | nal\_inet\_pton |
| Synopsis | nal\_status nal\_inet\_pton (nal\_int family, nal\_char\* src, void\* dst); |
| Description | Format the ASCII string into IP address. (Use big-endian on Nucleus) |
| Return Value | On success, *NAL\_SUCCESS* is returned; on error,”-1” or *NAL\_INVALID\_PARM* is returned. |

|  |  |
| --- | --- |
| Function Name | nal\_htons |
| Synopsis | nal\_uint16 nal\_htons (nal\_uint16 number); |
| Description | The function swaps 2 bytes of a 16-bit number from host order to network order and can work both on big-endian and little-endian. |
| Return Value | The byte-swapped input value is returned |

|  |  |
| --- | --- |
| Function Name | nal\_ntohs |
| Synopsis | nal\_uint16 nal\_ntohs (nal\_uint16 number); |
| Description | The function swaps 2 bytes of a 16-bit number from network order to host order and can work both on big-endian and little-endian. |
| Return Value | The byte-swapped input value is returned |

|  |  |
| --- | --- |
| Function Name | nal\_htonl |
| Synopsis | nal\_uint32 nal\_htonl (nal\_uint32 number); |
| Description | The function swaps 4 bytes of a long number from host order to network order and can work both on big-endian and little-endian. |
| Return Value | The byte-swapped input value is returned |

|  |  |
| --- | --- |
| Function Name | nal\_ntohl |
| Synopsis | nal\_uint32 nal\_ntohl (nal\_uint32 number); |
| Description | The function swaps 4 bytes of a long number from network order to host order and can work both on big-endian and little-endian. |
| Return Value | The byte-swapped input value is returned |

Error code mapping:

|  |  |  |
| --- | --- | --- |
| Function Name | Error Code | Cause |
| nal\_socket | NAL\_INVALID\_PROTOCOL | The input protocol is invalid |
|  | NAL\_NO\_SOCK\_MEMORY | No place in the socket structure (TCP\_MAX=30, UDP\_MAX=30) |
| nal\_bind | NAL\_INVALID\_PARM | Address is null or IP protocol is not IPv4 or IPv6. |
|  | NAL\_INVALID\_SOCKET | The socket ID is invalid or it has not been allocated before via nal\_socket(). |
|  | NAL\_INVALID\_PORT | IP address is not IP\_ADDR\_ANY and the port number has repeated. |
|  | NAL\_INVALID\_ADDRESS | The address cannot be found in the device or the format does not align to the IP protocol. |
| nal\_get\_sock\_name | NAL\_INVALID\_PARM | The input address is null or the input address length is invalid. |
|  | NAL\_INVALID\_SOCKET | The socket ID is invalid or it has not been allocated before via nal\_socket(). |
|  | NAL\_NOT\_CONNECTED | The socket ID doesn’t point to a valid socket. |
| nal\_connect | NAL\_INVALID\_PARM | The server address is null or IP type is not IPv4 or IPv6. |
|  | NAL\_INVALID\_SOCKET | Nal\_connect() has called before and both IP\_ADDR\_ANY and IPV6\_IS\_ADDR\_UNSPECIFIED have not set. |
|  | NAL\_NOT\_CONNECTED | Connecting to server fails or the server port does not exist. |
|  | NAL\_INVALID\_ADDRESS |  |
|  | NAL\_IS\_CONNECTING |  |
| nal\_is\_connected | NAL\_FALSE | The socket is not connected. |
|  | NAL\_INVALID\_SOCKET | The socket ID is not in the valid range. |
| nal\_send | NAL\_INVALID\_SOCKET | The IP protocol is neither TCP nor UDP with connected state. |
|  | NAL\_NOT\_CONNECTED | For TCP socket, it has not been connected and is not in the waiting or established state. |
|  | NAL\_WOULD\_BLOCK | For UDP socket, is non-blocking and the buffer is full at that time.  For TCP socket, it is non-blocking, the write channel is still open, but the buffer is full. |
|  | NAL\_INVALID\_PARM | Buffer is NULL but byte-to-send is not zero. |
|  | NAL\_NO\_DATA\_TRANSFER | For UDP socket, no data has been sent and the buffer is not full. |
|  | NAL\_DEVICE\_DOWN | For UDP socket, the device associated to the socket has gotten down. |
|  | NAL\_SOCKET\_CLOSED | For UDP socket, the socket has been closed. |
| nal\_recv | NAL\_NO\_PORT\_NUMBER | For TCP socket, there is no port number in the specific socket. |
|  | NAL\_INVALID\_SOCKET | The IP protocol is neither TCP nor UDP with connected state. |
|  | NAL\_INVALID\_OPTION | The NAL\_MSG\_PEEK flag is set without under UDP protocol. |
|  | NAL\_NOT\_CONNECTED | For TCP socket, the read side channel has been closed or no data is read without timeout with an error. |
|  | NAL\_WOULD\_BLOCK | For TCP socket, the read count is zero, the read channel is still alive and the socket is non-blocking.  For UDP socket, the read count is zero and the socket is non-blocking. |
|  | NAL\_NO\_DATA\_TRANSFER | For UDP socket, the read side channel has been closed. |
|  | NAL\_DEVICE\_DOWN | For UDP socket, the device associated to the socket has gotten down. |
| nal\_send\_to | NAL\_INVALID\_PARM | The port associated to the destination address exists. |
|  | NAL\_INVALID\_ADDRESS | The IP address is unspecified. |
|  | NAL\_NO\_DATA\_TRANSFER | The sent count is zero and the buffer is not full. |
|  | NAL\_WOULD\_BLOCK | There is no buffer and the socket is non-blocking. |
|  | NAL\_SOCKET\_CLOSED | The socket has been closed. |
| nal\_recv\_from | NAL\_INVALID\_PARM | The input remote address is NULL. |
|  | NAL\_WOULD\_BLOCK | The received count is zero, no error occurred, the read side channel is still alive and the socket is non-blocking. |
|  | NAL\_NO\_DATA\_TRANSFER | The read side channel has shut down. |
|  | NAL\_DEVICE\_DOWN | The device associated to the socket has gotten down. |
| nal\_getsockopt | NAL\_INVALID\_PARM | The “*optval*” is NULL or the NAL\_SO\_LINGER has set and “*optlen*” is not enough. |
|  | NAL\_INVALID\_OPTION | The input option name is invalid. |
|  | NAL\_INVALID\_LEVEL | The input “*level*” is invalid. |
|  | NAL\_INVAL | The input “*optname*” or “*optval*” is invalid. |
| nal\_setsockopt | NAL\_INVALID\_SOCKET | For NAL\_SO\_RCVBUF, the TCP port is invalid.  For NAL\_SO\_BINDTODEVICE, the socket is invalid.  For NAL\_SO\_AUTO\_CLEAR, the socket is invalid or it’s not a UDP socket. |
|  | NAL\_INVAL | The “*optval*” is NULL or “*optlen*” is shorter than structure size of the option. |
|  | NAL\_INVALID\_OPTION | The input option name is invalid. |
|  | NAL\_INVALID\_LEVEL | The input “*level*” is invalid. |
|  | NAL\_UNAVAILABLE | The Nucleus semaphore is not available now. |
|  | NAL\_INVALID\_PARM | The option value is NULL or option length is not as long as the type size. |
| nal\_inet\_ntop | NAL\_INVALID\_PARM | The input “family” is not NAL\_FAM\_IP or NAL\_FAM\_IP6 |
| nal\_inet\_pton | NAL\_INVALID\_PARM | The input “family” is not NAL\_FAM\_IP or NAL\_FAM\_IP6 |
| nal\_close\_socket | NAL\_INVALID\_SOCKET | The socket ID is invalid. |
|  | NAL\_NOT\_CONNECTED | The socket doesn’t point to a valid socket in internal socket structre. |
|  | -1 | TCP allocates event fail. |
|  | NAL\_NO\_PORT\_NUMBER | The TCP port number is invalid. |
|  | NAL\_INVAL | The “linger” option is invalid. |
|  | NAL\_SOCKET\_CLOSED | The socket which socket ID points to in not the same one as before. |
| nal\_shutdown | NAL\_INVALID\_SOCKET | The socket index is out of range. |
|  | NAL\_NOT\_CONNECTED | The socket index doesn’t point to a valid socket in the global socket structure. |
|  | NAL\_UNAVAILABLE | The Nucleus semaphore is not available now. |
|  | NAL\_INVALID\_PORT | The socket uses TCP protocol with *NAL\_SHUT\_WR* has chosen, and the port number is smaller than zero. |
| nal\_fd\_check | NAL\_TRUE | The bit that refers to the socket has been set. |
|  | NAL\_FALSE | Possible reasons:   1. The socket ID is out of range. 2. The “fd” input is NULL. 3. The bit that refers to the socket has not been set. |
| nal\_select | NAL\_NO\_SOCKETS | Possible reasons:   1. The input, “max\_sockets”, is zero. 2. The input, “max\_sockets” is larger than maximum socket number. 3. No bit referring to a specific socket has set. |
|  | NAL\_INVALID\_SOCKET |  |
|  | -1 |  |
|  | NAL\_NO\_DATA |  |
| nal\_fnctl |  |  |
|  |  |  |
| nal\_sendmsg |  |  |
|  |  |  |
| nal\_recvmsg |  |  |
|  |  |  |

*other IMCP error codes\*1* : IMCP is Internet Control Message Protocol used in the network layer for error reporting and router “signaling”; therefore, an IMCP error code would return when a IMCP packet is received by the socket. The following is the IMCP error code list:

NAL\_DEST\_UNREACH\_ADMIN

NAL\_DEST\_UNREACH\_ADDRESS

NAL\_DEST\_UNREACH\_PORT

NAL\_TIME\_EXCEED\_HOPLIMIT

NAL\_TIME\_EXCEED\_REASM

NAL\_PARM\_PROB\_HEADER

NAL\_PARM\_PROB\_NEXT\_HDR

NAL\_PARM\_PROB\_OPTION

NAL\_DEST\_UNREACH\_NET

NAL\_DEST\_UNREACH\_HOST

NAL\_DEST\_UNREACH\_PROT

NAL\_DEST\_UNREACH\_FRAG

NAL\_DEST\_UNREACH\_SRCFALL

NAL\_PARM\_PROB

NAL\_SOURCE\_QUENCH

*NAL Filter error code\*2*:

|  |  |
| --- | --- |
| Error Code | Cause |
| FIL\_IS\_REPEATED | The filter rule has set before |
| FIL\_INVALID\_PARM | Need to set NAL\_SO\_BINDTODEVICE first or the input filter ID is invalid |
| FIL\_INVALID\_SOCK | Invalid socket ID or invalid protocol or IP type is not IPv4 or IPv6 |
| FIL\_INVALID\_DEST | The remote address is not in IPv4 or IPv6 format. |
| FIL\_HAS\_DELETED | The filter rule has deleted already |
| FIL\_NEED\_BIND | Need to do nal\_bind() first |
| FIL\_RUN\_OUT | The filter quota has depleted |
| FIL\_ADD\_FAIL | Register NAL filter fail |
| FIL\_DEL\_FAIL | Deregister NAL filter fail |
| FIL\_INVALID\_PORT |  |

# Programming Guide for Network Abstraction Layer (NAL) Interface

Before looking into rules of usubg NAL interface, there are some concepts we want to introduce:

1. Since our device is in the LAN area, setting filters is a MUST to make sure data can be received correctly; therefore, we assume that you will have nal\_add\_filter() in the flow.
2. Most parts of socket programming here is the same as the usual one you use before. However, due to the reason in Item a&b, there are some rules you might need to be aware of and we will mention later.

In the begin, we are going to briefly introduce components we need later; then give a quick review to socket programming; in the end, give a whole view of using NAL interface.

## Components

### Clients

#### UDP Client

Since UDP clients are connectionless, nal\_connect() is not necessary to them. But nal\_bind() needs to be set for local address. They can do nal\_recv\_from() or nal\_send\_to() directly after getting the socket descriptor.

#### TCP Client

TCP clients are connection oriented; that is, they need to do nal\_bind() and nal\_connect() before doing nal\_send() or nal\_recv().

### Servers

#### UDP Server

Due to its connectionless character, a UDP server just acts like a UDP client.

#### TCP Server

TCP clients are connection oriented; therefore, a TCP server need to do nal\_listen() then nal\_accept() to deal with the connecting request from clients..

### NAL 5-tuple Filter

The NAL 5-tuple filter is used by IPcore to check if the packet is sent to the local device. Typically, the local device here is modem.

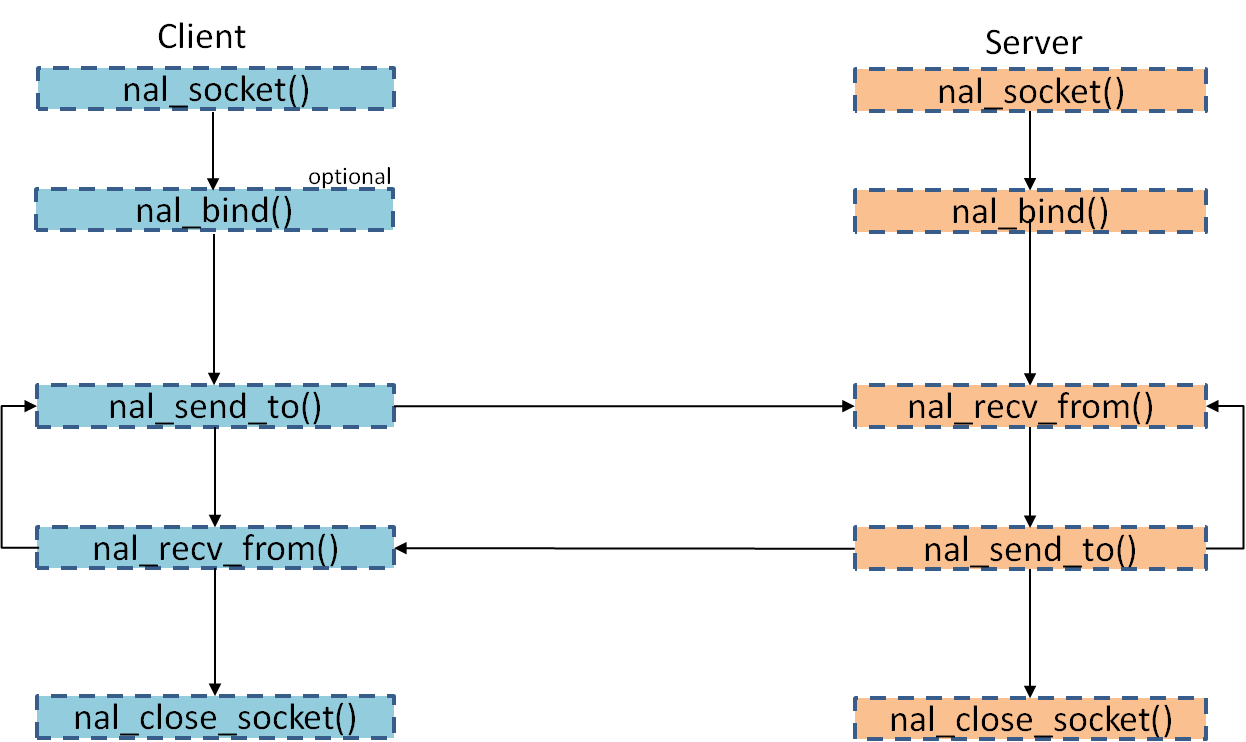
## Basic Data Transferring Flow

Let’s do some review about socket programming we learned before.

Do not think about NAL filter at this time, since it’s just a simple view to help you get into socket programming quickly, but if you are already an expert in it, just jump to the next paragraph.

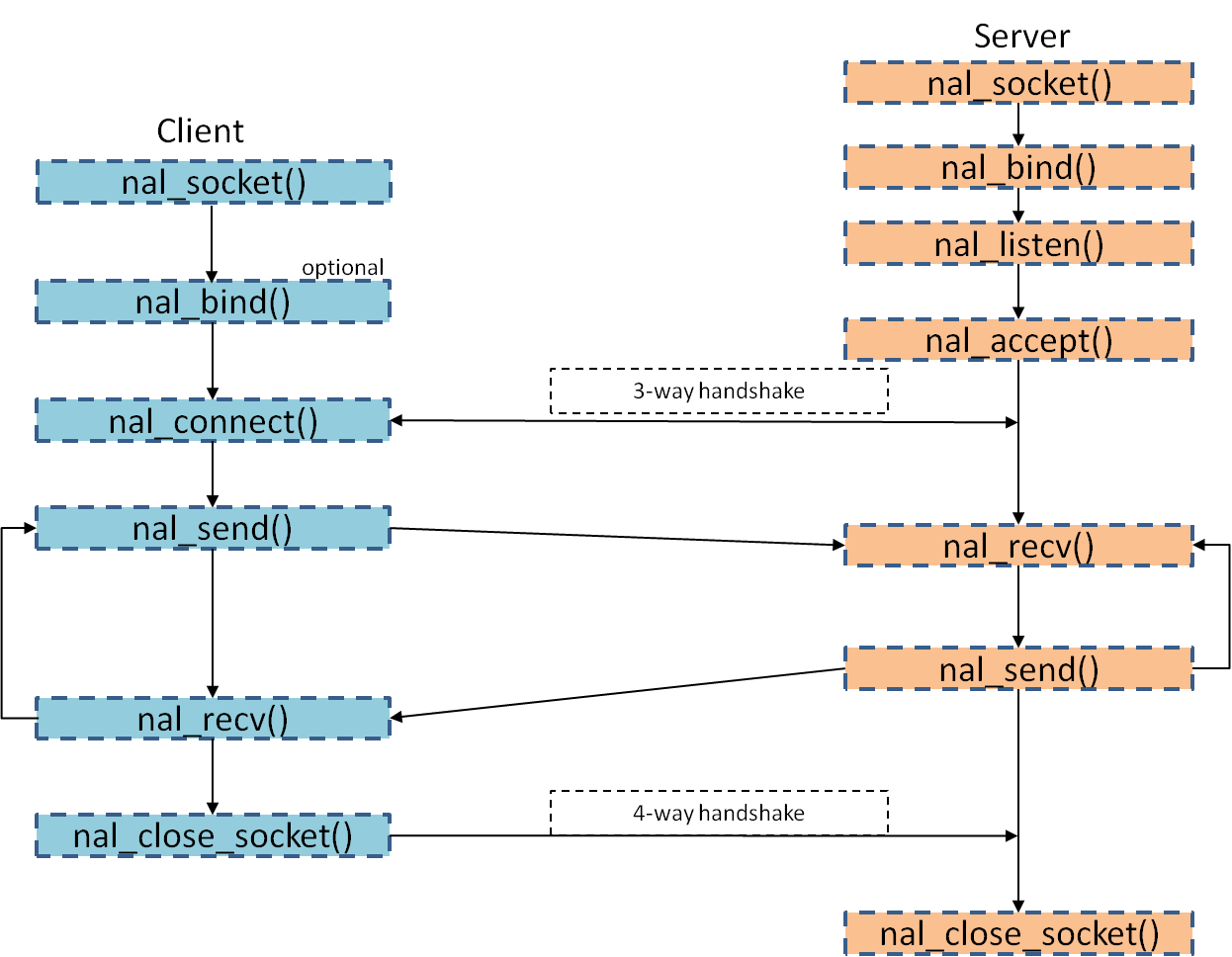
### Send/Receive between UDP sockets

As we learned before, the UDP client and UDP server do not need to do handshaking; that is, the UDP client just send data to server and meanwhile, server would block itself to receive data from clients until it receives data.



### Send/Receive between TCP sockets

Under TCP protocol, a client and a server need to establish a connection between each other and destroy the connection when one side wants to close the socket.



## Let’s start!

Now let’s talk about using NAL APIs here. Please bring the concept back that NAL filter is a MUST from now on.

Here are some rules you need to follow:

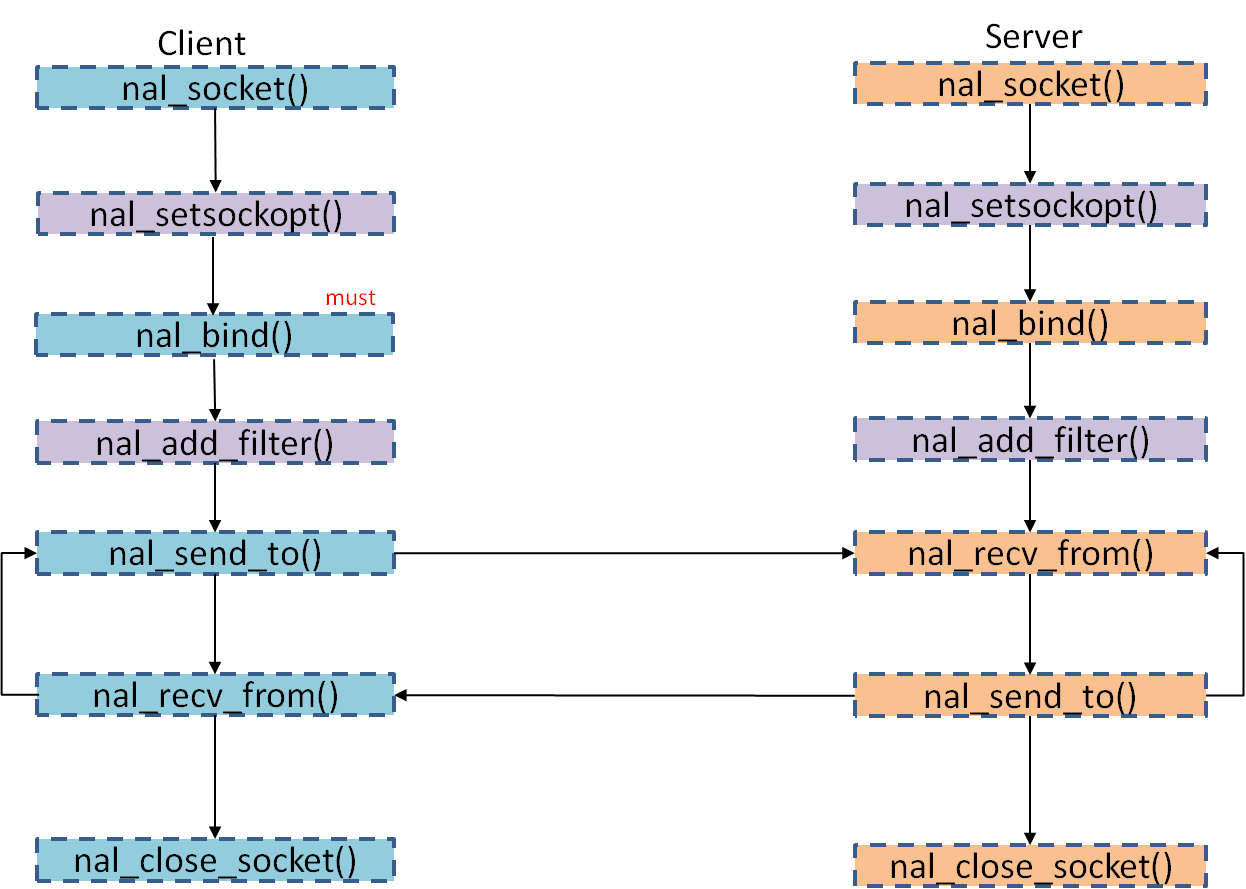
|  |  |  |
| --- | --- | --- |
| Index | Rules | Explanation |
| 1 | Always do nal\_bind() before *nal\_add\_filter(socketd, …)* in all TCP/UDP sockets. | After doing nal\_bind(), the local address and port number will be saved in the global socket structure, and the NAL filter can obtain INFO from the structure. |
| 2. | Use *nal\_add\_filter()* at least once for one socket *before* sending data or connecting to servers. | If NAL filter has not set, IPcore would not be aware of using filter and no data would come in. | |
| 3. | Set NAL\_SO\_LINGER (TCP only) & NAL\_SO\_BINDTODEVICE before setting nal\_add\_filter(). | 1. NAL\_SO\_LINGER: By default, the nal\_close\_socket() API returns without waiting for the TCP connection to close. That is, the socket descriptor may not be released, leading to problems if we create sockets right after nal\_close\_socket(). Therefore, set “NAL\_SO\_LINGER” can make the nal\_close\_socket() block until the whole TCP connection has been closed. 2. NAL\_SO\_BINDTODEVICE: Bind socket to the local device. Since there can be many devices in LAN, using this socket option can prevent data from sending to other devices. | |

More things about NAL filter that could be useful to you:

|  |  |
| --- | --- |
| Index | Statement |
| 1. | As mentioned before, *nal\_add\_filter(socketd, &otheraddr)* is used to add filter rules one by one; *nal\_add\_filter(socketd, NULL)* is used to filter in all data sent to the specific socket.  If *nal\_add\_filter(socketd, NULL)* has set, huge amount of packets are likely to burst into the local device, since each packet that sent to the local address matches the 3-tuple filter.  Therefore, use *nal\_add\_filter(socketd, NULL)* carefully. |
| 2. | If *nal\_shutdown(socketd, NAL\_SHUT\_RD or NAL\_SHUT\_RDWR)* or *nal\_close\_socket(socketd)* is called, the NAL interface would delete all filters associated with *socketd* automatically. |

The figure below shows the correct data transferring flow via NAL interface.

UDP data transferring flow



TCP data transferring flow



## Add Checking Mechanisms

In networking world, the connection status changes from time to time. Therefore, we need some checking mechanisms to do handlings.

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Goal | Mechanism | Return Value |
| 1. | A TCP client is connecting to a server. We want to know if the connection is still on the way to be established. | Use nal\_select() with “writefs”. | NAL\_NO\_DATA |
| 2. | A TCP client is connecting to a server. We want to know if the connection request has rejected by the server. | Use nal\_select() with “writefs”. | NAL\_INVALID\_SOCKET |
| 3. | Check if the client with non-blocking flag on has connected to server. | Use *nal\_is\_connected()* (Make sure the retry time is enough for the connection to be established.) | NAL\_TRUE |

HINT:

1. Set *timeout = 0* in nal\_select() is recommended to reduce the waiting time.
2. Do not use nal\_select() with “*readfs*” to distinguish the two situations mentioned above, since both case 1. and 2. return NAL\_NO\_DATA.
3. If nal\_is\_connected() returns true at first, but the server side crashes unexpectedly. The client side cannot detect the change, since no handshaking has down to close the connection.

## Sample Codes of NAL Socket Programming

### UDP client socket with NAL filter (part)

void nal\_test\_e\_entry(task\_entry\_struct \*task\_entry\_ptr)

{

nal\_status status;

/\*kal\_bool ret\_status = KAL\_TRUE;\*/

kal\_bool flag = KAL\_TRUE;

nal\_int32 udp\_sock, tcp\_sock;

nal\_addr\_struct myaddr, otheraddr;

nal\_addr\_struct myaddr2, otheraddr2;

nal\_char recvData[4096];

nal\_char sendData[4096];

nal\_int32 recvLen;

nal\_int16 otherAddrLen;

//nal\_int16 otherAddrLen2;

kal\_bool udpFail = KAL\_FALSE;

kal\_bool tcpFail = KAL\_FALSE;

/\*UDP Socket\*/

udp\_sock = nal\_socket(NAL\_FAMILY\_IP, NAL\_TYPE\_DGRAM, NAL\_NONE);

if (udp\_sock < 0) {

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): nal\_Socket failed");

while(1)

{

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): nal\_Socket failed");

}

flag = KAL\_FALSE;

goto start;

}

memset(&myaddr, 0, sizeof(nal\_addr\_struct));

myaddr.family = NAL\_FAMILY\_IP;

myaddr.port = nal\_htons(0x200);

nal\_inet\_pton(NAL\_FAM\_IP, "127.0.0.1", &(myaddr.id));

/\*UDP client bind\*/

status = nal\_bind(udp\_sock, &myaddr, 0);

if (status < 0) {

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): nal\_Bind failed %d.", status);

while(1)

{

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): UDP nal\_Bind failed %d.", status);

}

flag = KAL\_FALSE;

goto start;

}

//Register a NAL filter

nal\_add\_filter(udp\_sock, &otheraddr);

(Continue)

start:

while (1) {

if (flag == KAL\_FALSE) {

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): failed");

kal\_sleep\_task(kal\_milli\_secs\_to\_ticks(2000));

} else {

/\*UDP Socket: Send a message to server first\*/

nal\_char tmp[] = "I am a client.";

memcpy(sendData, tmp, sizeof(tmp));

if ((status = nal\_send\_to(udp\_sock, sendData, strlen(sendData), 0, (nal\_addr\_struct \*)&otheraddr, sizeof(nal\_addr\_struct))) < 0) {

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): nal\_Send\_To() failed, err = %d", status);

} else {

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): ST sent len = %d, sent data = %s.", status, sendData);

}

/\*UDP Socket: Receive\*/

udpFail = KAL\_FALSE;

tcpFail = KAL\_FALSE;

recvLen = nal\_recv\_from(udp\_sock, recvData, sizeof(recvData), 0, (nal\_addr\_struct \*)&otheraddr, &otherAddrLen);

if (recvLen > 0) {

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): nal\_Recv\_From() recvData = %s.", recvData);

} else {

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): nal\_Recv\_From() failed, err = %d", recvLen);

udpFail = KAL\_TRUE;

}

}

}

(End)

### TCP client socket with NAL filter (part)

void nal\_test\_e\_entry(task\_entry\_struct \*task\_entry\_ptr)

{

nal\_status status;

/\*kal\_bool ret\_status = KAL\_TRUE;\*/

kal\_bool flag = KAL\_TRUE;

nal\_int32 udp\_sock, tcp\_sock;

nal\_addr\_struct myaddr, otheraddr;

nal\_addr\_struct myaddr2, otheraddr2;

nal\_char recvData[4096];

nal\_char sendData[4096];

nal\_int32 recvLen;

nal\_int16 otherAddrLen;

//nal\_int16 otherAddrLen2;

kal\_bool udpFail = KAL\_FALSE;

kal\_bool tcpFail = KAL\_FALSE;

/\*TCP Socket\*/

tcp\_sock = nal\_socket(NAL\_FAMILY\_IP, NAL\_TYPE\_STREAM, NAL\_NONE);

if (tcp\_sock < 0) {

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): nal\_Socket failed");

while(1)

{

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): nal\_Socket failed");

}

flag = KAL\_FALSE;

goto start;

}

/\*TCP Connect\*/

memset(&myaddr2, 0, sizeof(nal\_addr\_struct));

myaddr2.family = NAL\_FAMILY\_IP;

myaddr2.port = nal\_htons(0x888);

nal\_inet\_pton(NAL\_FAM\_IP, "127.0.0.1", &(myaddr2.id));

memset(&otheraddr2, 0, sizeof(nal\_addr\_struct));

otheraddr2.family = NAL\_FAMILY\_IP;

otheraddr2.port = nal\_htons(0x1234);

nal\_inet\_pton(NAL\_FAM\_IP, "127.0.0.1", &(otheraddr2.id));

ASSERT(SCK\_Sockets[tcp\_sock]->s\_port\_index >= 0);

status = nal\_bind(tcp\_sock, &myaddr2, 0);

ASSERT(SCK\_Sockets[tcp\_sock]->s\_port\_index >= 0);

if (status < 0) {

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): nal\_Bind failed %d.", status);

flag = KAL\_FALSE;

goto start;

}

nal\_fcntl(tcp\_sock, NAL\_SETFLAG, NAL\_NO\_BLOCK);

kal\_sleep\_task(kal\_milli\_secs\_to\_ticks(5000));

status = nal\_connect(tcp\_sock, &otheraddr2, sizeof(nal\_addr\_struct));

do{

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): waiting nal\_Is\_Connected status = %d.", status);

kal\_sleep\_task(kal\_milli\_secs\_to\_ticks(2000));

} while ((nal\_is\_connected(tcp\_sock) == KAL\_FALSE));

start:

while (1) {

if (flag == KAL\_FALSE) {

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): failed");

kal\_sleep\_task(kal\_milli\_secs\_to\_ticks(2000));

} else {

/\*TCP Socket: Receive\*/

recvLen = nal\_recv(tcp\_sock, recvData, sizeof(recvData), 0);

if (recvLen > 0) {

//Receive the data successfully

} else {

kal\_prompt\_trace(MOD\_NIL, "nal\_test\_e\_entry(): nal\_Recv() failed, err = %d", recvLen);

tcpFail = KAL\_TRUE;

}

if (udpFail == KAL\_TRUE && tcpFail == KAL\_TRUE) {

kal\_sleep\_task(kal\_milli\_secs\_to\_ticks(2000));

}

}

} /\* while (1)\*/

}