Network Functions

htonl(), htons(), ntohl(), ntohs()

SUMMARY Convert values between host and network byte order

SYNOPSIS **#import <netinet/in.h>**

u_long **htonl(**u_long *hostlong***)**
u_short **htons(**u_short *hostshort***)**
u_long **ntohl(**u_long *netlong***)**
u_short **ntohs(**u_short *netshort***)**

DESCRIPTION These functions and macros simulate the C library functions of the same name. See the UNIX manual page for **byteorder** for more information.

if_attach()

SUMMARY Initialize and install a new netif

SYNOPSIS **#import <net/netif.h>**

netif_t **if_attach(**if_init_func_t *init_func*, if_input_func_t *input_func*, if_output_func_t€*output_func*, if_getbuf_func_t *getbuf_func*, if_control_func_t€*control_func*, const char **name*, unsigned int *unit*, const€char€**type*, unsigned int *mtu*, unsigned int *flags*, netif_class_t *class*, void€**private***)*

ARGUMENTS *init_func*: The initialization function of this module.

input_func: The input function of this module.

output_func: The output function of this module.

getbuf_func: The buffer allocation function of this module.

control_func: The control function of this module.

name: A constant string that names this module (for example, ªenº).

unit: The unit number of this module (for example, **0**).

type: A constant string that describes the type of this module (for example, ª10MB Ethernetº).

mtu: The maximum transfer unit (for example, **1500** for Ethernet). This is the maximum amount of data your module can send or receive. Note that protocol-level modules must return the minimum of either the protocol limit or the network device driver's limit (minus header information).

flags: Initial flags for the interface. Possible values are:

IFF_UP:	If true, this interface is working.
IFF_BROADCAST:	If true, this interface supports broadcast.
IFF_LOOPBACK:	If true, this interface is local only.
IFF_POINTTOPOINT:	If true, this is a point-to-point interface.

class: The class of this interface. Possible values are:

NETIFCLASS_REAL:	Network driver (a module that manipulates the hardware).
NETIFCLASS_VIRTUAL:	Protocol handler (a module that doesn't talk to hardware).
NETIFCLASS_SNIFFER:	Packet sniffer (a module that merely examines packets).

private: Private data, which can be retrieved using **if_private()**.

DESCRIPTION Initializes a new netif and installs it, returning the resulting netif. Network device drivers should call this function at load time. Protocol handlers and packet sniffers should instead call **if_registervirtual()** at load time, supplying as an argument the callback function that calls **if_attach()**.

The first five arguments are the functions that are associated with netifs. These functions are described in Chapter 8, ªNetwork Modules.º

This function doesn't check any of its arguments, so it always succeeds.

EXAMPLE

```
ifp = if_attach(NULL, myhandler_input, myhandler_output,
    myhandler_getbuf, myhandler_control, name, unit, IFTYPE_IP,
    MYMTU, IFF_BROADCAST, NETIFCLASS_VIRTUAL, ifprivate);
```

SEE ALSO **if_flags()**, **if_output()**, **if_init()**, **if_control()**, **if_ioctl()**, **if_getbuf()**, **if_handle_input()**

if_class()

SUMMARY Get the class of a netif

SYNOPSIS **#import <net/netif.h>**

netif_class_t **if_class(**netif_t *netif***)**

ARGUMENTS *netif*: The module to get the class for.

DESCRIPTION This function returns the class of *netif*. The value returned is one of the following:

- NETIFCLASS_REAL
- NETIFCLASS_VIRTUAL
- NETIFCLASS_SNIFFER

EXAMPLE

```
netif_t ni;

for (ni = iflist_first(); ni != NULL; ni = iflist_next(ni)) {
    printf("name: %s%d, class: %s\n", if_name(ni), if_unit(ni),
        (if_class(ni) == NETIFCLASS_REAL) ? "real" :
            ((if_class(ni) == NETIFCLASS_VIRTUAL) ? "virtual" :
            ((if_class(ni) == NETIFCLASS_SNIFFER) ? "sniffer" : "unknown")));
}
```

SEE ALSO **iflist_first()**, **iflist_next()**

if_collisions(), if_collisions_set()

SUMMARY Get or set the number of collisions

SYNOPSIS **#import <net/netif.h>**

unsigned int **if_collisions(**netif_t *netif***)**
void **if_collisions_set(**netif_t *netif*, unsigned int *collisions***)**

ARGUMENTS *netif*: The module to get or set collision data for.

collisions: The new number of collisions.

DESCRIPTION The functions **if_collisions()** and **if_collisions_set()** get and set, respectively, the number of collisions encountered. Only the module corresponding to *netif* should call **if_collisions_set()**.

EXAMPLE
```
printf("Number of collisions encountered so far:  %d\n",
    if_collisions(netif));
```

if_control(), if_getbuf(), if_init(), if_ioctl(), if_output()

SUMMARY Call one of the functions associated with a netif

SYNOPSIS **#import <net/netif.h>**

int **if_control(**netif_t *netif*, const char **command*, void **data***)**
netbuf_t **if_getbuf(**netif_t *netif***)**
int **if_init(**netif_t *netif***)**
int **if_ioctl(**netif_t *netif*, unsigned int *command*, void **data***)**
int **if_output(**netif_t *netif*, netbuf_t *packet*, void **address***)**

ARGUMENTS *netif*: The module whose function should be called.

command: The control command that should be executed.

data: Data specific to the control command.

packet: The packet that should be passed to *netif*'s output function.

address: The address that should be specified in the call to *netif*'s output function.

DESCRIPTION The **if_control()** function calls *netif*'s control function, **if_getbuf()** calls *netif*'s buffer allocation function, **if_init()** calls *netif*'s initialization function, and **if_output()** calls *netif*'s output function.

It is recommended that you don't use **if_ioctl()**; it's provided only for compatibility with UNIX code that operates using **ioctl**. The **if_ioctl()** function calls *netif*'s control function.

Except for **if_getbuf()**, these functions return ENXIO if the corresponding function isn't implemented in *netif*; otherwise they return the value returned by the call to the corresponding function. The **if_getbuf()** function returns NULL if the corresponding function isn't implemented.

EXAMPLE
```
nb = if_getbuf(ifp);
if (nb == NULL)
    return ENOBUFS;
```

SEE ALSO **if_attach()**

if_detach()

SUMMARY Detach a netif so it can be unloaded

SYNOPSIS **#import <net/netif.h>**

void **if_detach(**netif_t *netif***)**

ARGUMENTS *netif*: The module to detach.

DESCRIPTION The **if_detach()** function removes *netif* from the system lists of attached netifs. You should call this function before unloading a network module.

EXAMPLE

```
netif_t ni;

for (ni = iflist_first(); ni != NULL; ni = iflist_next(ni)) {
    if (if_unit(ni) == myunit && if_type(ni) == mytype &&
        strcmp(if_name(ni), myname) == 0) {
            if_detach(ni);
            break;
    }
}
```

if_flags(), if_flags_set()

SUMMARY Get or set the flags associated with a netif

SYNOPSIS **#import <net/netif.h>**

unsigned int **if_flags(**netif_t *netif***)**
void **if_flags_set(**netif_t *netif*, unsigned int *flags***)**

ARGUMENTS *netif*: The module to get or set flags for.

flags: The new flags.

DESCRIPTION The functions **if_flags()** and **if_flags_set()** get and set, respectively, the flags associated with *netif*. Only the module corresponding to *netif* should use **if_flags_set()**.

Possible flag values are:

IFF_UP:	If true, this interface is working.
IFF_BROADCAST:	If true, this interface supports broadcast.
IFF_LOOPBACK:	If true, this interface is local only.
IFF_POINTTOPOINT:	If true, this is a point-to-point interface.

EXAMPLE

```
if_flags_set(ifp, if_flags(ifp) | IFF_UP);
```

SEE ALSO **if_attach()**

if_handle_input()

SUMMARY Dispatch an input packet to a protocol handler

SYNOPSIS **#import <net/netif.h>**

int **if_handle_input(**netif_t *netif*, netbuf_t *packet*, void **extra***)*

ARGUMENTS *netif*: This module, which must be a network device driver.

packet: The input packet.

extra: Any extra data that might be needed by the protocol handler.

DESCRIPTION Call this in a network device driver to have an input packet dispatched to a protocol handler. This

function calls the input functions of one or more protocol handlers, passing along the *packet* and *extra* arguments.

This function returns EAFNOSUPPORT if no protocol handler accepts the packet.

EXAMPLE

```
if (nb == 0) {
    printf ("Error:  buffer is null\n");
    goto resetup;
}
else {
    if_handle_input(netif, nb, NULL);
}
```

if_ierrors(), if_ierrors_set(), if_oerrors(), if_oerrors_set()

SUMMARY Get or set the number of input or output errors

SYNOPSIS **#import <net/netif.h>**

unsigned int **if_ierrors(**netif_t *netif***)**
void **if_ierrors_set(**netif_t *netif*, unsigned int *ierrors***)**
unsigned int **if_oerrors(**netif_t *netif***)**
void **if_oerrors_set(**netif_t *netif*, unsigned int *oerrors***)**

ARGUMENTS *netif*: The module for which to access the number of errors.

ierrors: The number of input errors.

oerrors: The number of output errors.

DESCRIPTION The functions **if_ierrors()** and **if_ierrors_set()** get and set, respectively, the number of€input errors encountered. Only the module corresponding to *netif* should call **if_ierrors_set().**

The **if_oerrors()** function and the **if_oerrors_set()** function get and set the number of output errors encountered. Again, only the *netif*'s module should call **if_oerrors_set()**.

EXAMPLE

```
error = if_output(lowernetif, nb, (void *)addr);
if (error == 0)
    if_opackets_set(netif, if_opackets(netif) + 1);
else
    if_oerrors_set(netif, if_oerrors(netif) + 1);
return (error);
```

if_ipackets(), if_ipackets_set(), if_opackets(), if_opackets_set()

SUMMARY Get or set the number of packets received or sent

SYNOPSIS **#import <net/netif.h>**

unsigned int **if_ipackets(**netif_t *netif***)**
void **if_ipackets_set(**netif_t *netif*, unsigned int *ipackets***)**
unsigned int **if_opackets(**netif_t *netif***)**
void **if_opackets_set(**netif_t *netif*, unsigned int *opackets***)**

ARGUMENTS *netif*: The module whose packet information is to be accessed.

ipackets: The number of input packets handled by this module since it was loaded.

opackets: The number of packets sent to a lower level by this module since it was loaded.

DESCRIPTION The **if_ipackets()** function gets and the **if_ipackets_set()** function sets the number of input packets handled. Only the module specified by *netif* should call **if_ipackets_set()**.

Similarly, **if_opackets()** gets and **if_opackets_set()** sets the number of output packets sent. Only *netif*'s module should call **if_opackets_set()**.

EXAMPLE

```
error = if_output(lowernetif, nb, (void *)addr);
if (error == 0) {
    if_opackets_set(netif, if_opackets(netif) + 1);
} else {
    if_oerrors_set(netif, if_oerrors(netif) + 1);
}
return (error);
```

if_mtu(), if_name(), if_private(), if_type(), if_unit()

SUMMARY Get information about a netif

SYNOPSIS **#import <net/netif.h>**

unsigned int **if_mtu(**netif_t *netif***)**
const char ***if_name(**netif_t *netif***)**
void ***if_private(**netif_t *netif***)**
const char ***if_type(**netif_t *netif***)**
unsigned int **if_unit(**netif_t *netif***)**

ARGUMENTS *netif*: The netif whose data is being requested.

DESCRIPTION These functions return the following information about *netif*:

if_mtu():	Its maximum transfer unit (for example, **1500** for Ethernet).
if_name():	Its name (for example, ªenº).
if_private():	Its private data.
if_type():	Its type string (for example, ª10MB Ethernetº).
if_unit():	Its unit number (for example, **0**).

See Chapter 8 for more information on netifs. Only the module specified by *netif* should call **if_private()**.

EXAMPLE

```
((venip_private_t *)if_private(netif))->lowernetif = realnetif;
```

SEE ALSO **if_flags()**, **if_flags_set()**, **if_attach()**

if_registervirtual()

SUMMARY Register a callback function that calls **if_attach()**

SYNOPSIS **#import <net/netif.h>**

void **if_registervirtual(**if_attach_func_t *attach_func*, void **private***)**

ARGUMENTS *attach_func*: The callback function.

private: Data to be passed to the callback function.

DESCRIPTION For use by protocol handlers and packet sniffers. This function registers a callback function that should call **if_attach()**. See Chapter 8 for more information on implementing this callback function.

EXAMPLE `if_registervirtual(myhandler_attach, NULL);`

iflist_first(), iflist_next()

SUMMARY Find the first or next netif in the system netif list

SYNOPSIS **#import <net/netif.h>**

netif_t **iflist_first(**void**)**
netif_t **iflist_next(**netif_t *netif***)**

ARGUMENTS *netif*: The previous netif in the list.

DESCRIPTION These functions are useful for cycling through the system list of attached netifs. The **iflist_first()** function returns the first netif in the list. The **iflist_next()** function returns the netif after *netif* in the list.

EXAMPLE

```
netif_t ni;

for (ni = iflist_first(); ni != NULL; ni = iflist_next(ni)) {
    printf("name: %s%d, type: %s\n", if_name(ni), if_type(ni));
}
```

inet_queue()

SUMMARY Give an IP input packet to the kernel for processing

SYNOPSIS **#import <net/netif.h>**
#import <net/netbuf.h>

void **inet_queue(**netif_t *netif*, netbuf_t *netbuf***)**

ARGUMENTS *netif*: The protocol handler.

netbuf: The packet to hand over.

DESCRIPTION IP protocol handlers wishing to hand over their input packets to the kernel for processing should call this function.

You can safely call this function from an interrupt handler, since it doesn't block.

EXAMPLE

```
nb_shrink_top(netbuf, HDRSIZE);
if_ipackets_set(netif, if_ipackets(netif) + 1);
inet_queue(netif, netbuf);
```

nb_alloc(), nb_alloc_wrapper()

SUMMARY Allocate a netbuf or a netbuf wrapper

SYNOPSIS **#import <net/netbuf.h>**

netbuf_t **nb_alloc(**unsigned int *size***)**
netbuf_t **nb_alloc_wrapper(**void **data*, unsigned int *size*, void€*freefunc*(void€*), void€**freefunc_arg***)**

ARGUMENTS *size????*: The size of the data to be stored.

data: The data to be stored.

freefunc: The function to be called when the netbuf is freed.

freefunc_arg: The argument to be passed to *freefunc*.

DESCRIPTION The **nb_alloc()** function allocates a netbuf containing *size* bytes. It returns the netbuf.

The **nb_alloc_wrapper()** function allocates only a wrapper for a netbuf. Use this function when you have already allocated space for the packet. The returned netbuf's original start of data is *data*, and its bottom pointer is initialized to *data* + *size* -1. If **nb_free()** is called on the returned netbuf, *freefunc* will be called with the argument *freefunc_arg*. The value of *freefunc* can't be null.

Note: Don't call either **nb_alloc()** or **nb_alloc_wrapper()** from an interrupt handler. (Both functions can sleep.)

Both **nb_alloc()** and **nb_alloc_wrapper()** return a null pointer if they fail.

EXAMPLE
```
nb = nb_alloc_wrapper((void *)buf, HDR_SIZE + MYMTU,
    mydriver_buf_put, (void *)buf);
```

SEE ALSO **nb_free()**, **nb_free_wrapper()**

nb_free(), nb_free_wrapper()

SUMMARY Free a netbuf or only its wrapper

SYNOPSIS **#import <net/netbuf.h>**

void **nb_free(**netbuf_t *nb***)**
void **nb_free_wrapper(**netbuf_t *nb***)**

ARGUMENTS *nb*: The netbuf to be freed (or whose wrapper is to be freed).

DESCRIPTION The **nb_free()** function frees the netbuf *nb*. The **nb_free_wrapper()** function frees only the wrapper of *nb*, leaving its data area intact.

EXAMPLE
```
nb_free_wrapper(nb);
```

SEE ALSO **nb_alloc()**, **nb_alloc_wrapper()**

nb_grow_bot(), nb_shrink_bot(), nb_grow_top(), nb_shrink_top()

SUMMARY Change the size of a netbuf

SYNOPSIS **#import <net/netbuf.h>**

int **nb_grow_bot(**netbuf_t *nb*, unsigned int *size***)**
int **nb_shrink_bot(**netbuf_t *nb*, unsigned int *size***)**
int **nb_grow_top(**netbuf_t *nb*, unsigned int *size***)**
int **nb_shrink_top(**netbuf_t *nb*, unsigned int *size***)**

ARGUMENTS *nb*: The netbuf to be affected.

size: The number of bytes to add or delete from the top or bottom pointer.

DESCRIPTION The **nb_grow_bot()** function moves the bottom pointer down. After this call, the data is assumed to end *size* bytes after where it used to end (the data area has effectively grown).

The **nb_shrink_bot()** function moves the bottom pointer up, effectively shrinking the data area. After this call, the data is assumed to end *size* bytes before where it used to end.

The **nb_grow_top()** function moves the top pointer up, enlarging the data area. After this call, the data is assumed to start *size* bytes before where it used to start.

The **nb_shrink_top()** function moves the top pointer down, shrinking the data area. After this call, the data is assumed to start *size* bytes beyond where it used to start.

These functions perform no error checking, so they always succeed and return zero.

Warning: Writing to space outside of the original starting and ending points will cause serious errors, since the extra memory doesn't belong to the netbuf data section.

EXAMPLE

```
nb_shrink_top(netbuf, HDRSIZE);
if_ipackets_set(netif, if_ipackets(netif) + 1);
inet_queue(netif, netbuf);
```

SEE ALSO **nb_size()**

nb_map()

SUMMARY Get a pointer to the data stored in a netbuf

SYNOPSIS **#import <net/netbuf.h>**

char ***nb_map(**netbuf_t *nb***)**

ARGUMENTS *nb*: The netbuf whose data you want.

DESCRIPTION Returns a pointer to the data stored in *nb*. The pointer is valid only until another **nb_*()** function is called on *nb*.

This function returns a null pointer if it fails.

EXAMPLE

```
char *map = nb_map(nb);
```

SEE ALSO **nb_read()**, **nb_write()**, **nb_size()**

nb_read(), nb_write()

SUMMARY Access data in a netbuf

SYNOPSIS **#import <net/netbuf.h>**

int **nb_read(**netbuf_t *nb*, unsigned int *offset*, unsigned int *size*, void **target***)**
int **nb_write(**netbuf_t *nb*, unsigned int *offset*, unsigned int *size*, void **source***)*

ARGUMENTS *nb*: The netbuf whose data you want to read or write.

offset: The offset of the start of data from the beginning of the netbuf.

size: The number of bytes to be read or written.

target: The place to put the data.

source: The place to read the data from.

DESCRIPTION The **nb_read()** function reads data from *nb* into *target*. It starts at offset *offset* from the starting point in the data and reads *size* bytes into *target*.

The **nb_write()** function writes data into *nb*. It starts writing at offset *offset* from the starting point in the data and writes *size* bytes from *source*.

Both **nb_read()** and **nb_write()** return zero if the call was successful; otherwise, they return a nonzero value.

EXAMPLE

```
char    buf[MAXTRAILERBUF];

/*
 * Save copy of data.
 */
nb_read(nb, HDRSIZE, offset, &buf);
```

SEE ALSO **nb_map()**

nb_size()

SUMMARY Get the size of the data stored in a netbuf

SYNOPSIS **#import <net/netbuf.h>**

unsigned int **nb_size(**netbuf_t *nb***)**

ARGUMENTS *nb*: The netbuf to get the size of.

DESCRIPTION Returns the size (in bytes) of the data stored by *nb*.

EXAMPLE

```
if (HDRSIZE + offset + size > nb_size(nb)) {
    return (EAFNOSUPPORT);
}
```

SEE ALSO **nb_grow_bot()**, **nb_shrink_bot()**, **nb_grow_top()**, **nb_shrink_top()**, **nb_map()**