



# Summary of Kernel Support Functions

This appendix summarizes the kernel support functions (and some macros that behave like functions) that loadable kernel servers can call. Within the general categories of "General Functions" and "Network Functions," function declarations are further subgrouped to help you identify their interrelationships.

Chapter 10, "Kernel Support Functions," contains full descriptions of all the functions listed here. In addition, loadable kernel servers can use many Mach kernel functions, which are described in a section of Chapter 4, "Mach Functions." The Mach kernel functions are summarized in the manual, *NeXTSTEP Programmer Interface Summary*.

## General Functions

This section contains a summary of the general purpose kernel support functions. Most of the functions and macros in this section are declared through either the **kernserv/kern\_server\_types.h** or **kernserv/prototypes.h** header file.

## Time Functions

Busy-wait for a certain amount of time:

```
void          DELAY(unsigned int usecs)
```

Get or set the current time:

```
ns_time_t    clock_value(clock_types_t which_clock)
void         set_clock(clock_types_t which_clock, ns_time_t ns)
```

Get information about a clock:

```
chrono_attributes_t
              clock_attributes(clock_types_t which_clock)
```

Convert between **ns\_time\_t** and **timeval** data formats:

```
void          ns_time_to_timeval(ns_time_t ns, struct timeval *tv)
ns_time_t     timeval_to_ns_time(struct timeval *tv)
```

Schedule or unschedule a function to be called later:

```
void          ns_abstimeout(func function, vm_address_t arg, ns_time_t deadline, int priority)
void          ns_timeout(func function, vm_address_t arg, ns_time_t time, int priority)
boolean_t     ns_untimeout(func function, vm_address_t arg)
```

# Memory Functions

Make addresses pageable or memory-resident:

kern_return_t	<b>kern_serv_unwire_range</b> (kern_server_t * <i>ksp</i> , vm_address_t <i>address</i> , vm_size_t <i>size</i> )
kern_return_t	<b>kern_serv_wire_range</b> (kern_server_t * <i>ksp</i> , vm_address_t <i>address</i> , vm_size_t <i>size</i> )

Copy or initialize data:

void	<b>bcopy</b> (void * <i>from</i> , void * <i>to</i> , int <i>length</i> )
void	<b>bytecopy</b> (void * <i>from</i> , void * <i>to</i> , int <i>length</i> )
void	<b>bzero</b> (void * <i>address</i> , int <i>length</i> )

Allocate or free memory:

void *	<b>kalloc</b> (int <i>size</i> )
void	<b>kfree</b> (void * <i>address</i> , int <i>size</i> )
void *	<b>kget</b> (int <i>size</i> )

# Critical Section and Synchronization Functions

Use read and write locks:

lock_t	<b>lock_alloc</b> (void)
void	<b>lock_free</b> (lock_t <i>lock</i> )
void	<b>lock_done</b> (lock_t <i>lock</i> )
void	<b>lock_init</b> (lock_t <i>lock</i> , boolean_t <i>can_sleep</i> )
void	<b>lock_read</b> (lock_t <i>lock</i> )
void	<b>lock_write</b> (lock_t <i>lock</i> )

Use simple, nonsleeping locks:

void	<b>simple_lock</b> (simple_lock_t <i>lock</i> )
simple_lock_t	<b>simple_lock_alloc</b> (void)
void	<b>simple_lock_free</b> (simple_lock_t <i>lock</i> )
void	<b>simple_lock_init</b> (simple_lock_t <i>lock</i> )
void	<b>simple_unlock</b> (simple_lock_t <i>lock</i> )

Cause a thread to sleep or wake up:

void	<b>assert_wait</b> (int <i>event</i> , boolean_t <i>interruptible</i> )
void	<b>clear_wait</b> (thread_t <i>thread</i> , int <i>result</i> , boolean_t <i>interrupt_only</i> )
void	<b>thread_block</b> (void)
void	<b>thread_set_timeout</b> (int <i>ticks</i> )
void	<b>thread_sleep</b> (int <i>event</i> , simple_lock_t <i>lock</i> , boolean_t <i>interruptible</i> )
void	<b>thread_wakeup</b> (int <i>event</i> )

# General Task and Thread Functions

Get information about this thread or task:

task_t	<b>current_task</b> (void)
int	<b>thread_wait_result</b> (void)

Create or kill a thread:

thread_t	<b>kernel_thread</b> (task_t <i>task</i> , void (* <i>start</i> )(void))
void	<b>thread_halt_self</b> (void)

# Port and Message Functions

Request notification messages, such as port death notification:

kern_return_t	<b>kern_serv_notify</b> (kern_server_t * <i>ksp</i> , port_t <i>reply_port</i> , port_t <i>request_port</i> )
---------------	---

Get the kernel's task port:

port_t	<b>kern_serv_kernel_task_port</b> (void)
--------	--

Get or set information about this server's ports:

port_t	<b>kern_serv_bootstrap_port</b> (kern_server_t * <i>ksp</i> )
port_t	<b>kern_serv_local_port</b> (kern_server_t * <i>ksp</i> )
port_t	<b>kern_serv_notify_port</b> (kern_server_t * <i>ksp</i> )
void	<b>kern_serv_port_gone</b> (kern_server_t * <i>ksp</i> , port_name_t <i>port</i> )
kern_return_t	<b>kern_serv_port_proc</b> (kern_server_t * <i>ksp</i> , port_all_t <i>port</i> , port_map_proc_t <i>function</i> , int <i>arg</i> )
kern_return_t	<b>kern_serv_port_serv</b> (kern_server_t * <i>ksp</i> , port_all_t <i>port</i> , port_map_proc_t <i>function</i> , int <i>arg</i> )
port_set_name_t	<b>kern_serv_port_set</b> (kern_server_t * <i>ksp</i> )

# Hardware Interface Functions

Set up or remove an interrupt handler:

int	<b>install_polled_intr</b> (int <i>which</i> , int (* <i>my_intr</i> )(void))
int	<b>uninstall_polled_intr</b> (int <i>which</i> , int (* <i>my_intr</i> )(void))

Get or test a virtual address that corresponds to a hardware address:

caddr_t	<b>map_addr</b> (caddr_t <i>address</i> , int <i>size</i> )
int	<b>probe_rb</b> (void * <i>address</i> )

Change or determine the processor level:

int	<b>curipl</b> (void)
int	<b>spl0</b> (void), <b>spl1</b> (void), <b>spl2</b> (void), <b>spl3</b> (void), <b>spl4</b> (void), <b>spl5</b> (void), <b>spl6</b> (void), <b>spl7</b> (void)
void	<b>splx</b> (int <i>priority</i> )

# Logging and Debugging Functions

Kill the loadable kernel server:

void	<b>ASSERT</b> (int <i>expression</i> )
kern_return_t	<b>kern_serv_panic</b> (port_t <i>bootstrap_port</i> , panic_msg_t <i>message</i> )
void	<b>panic</b> (char * <i>string</i> )

Log a message:

void	<b>kern_serv_log</b> (kern_server_t * <i>ksp</i> , int <i>log_level</i> , char * <i>format</i> , <i>arg1</i> , ..., <i>arg5</i> )
int	<b>log</b> (int <i>level</i> , char * <i>format</i> , <i>arg</i> , ...)
int	<b>printf</b> (char * <i>format</i> , <i>arg</i> , ...)

# UNIX Support Functions

In a UNIX-style server, determine whether the user has root privileges:

```
int          suser(void)
```

In a UNIX-style server, wait for I/O completion on a buffer:

```
void          biodone(struct buf *bp)
void          biowait(struct buf *bp)
```

In a UNIX-style server, copy data between user and kernel address space:

```
int           copyin(void *from, void *to, int length)
int           copyout(void *from, void *to, int length)
```

In a UNIX-style server, implement the **select()** system call:

```
int           selthreadcache(void **waiterPtr)
void          selthreadclear(void **waiterPtr)
int           selwakeup(void *waiter, int collided)
```

## Miscellaneous Functions

Modify or inspect a string:

```
int           sprintf(char *string, char *format, arg, ...)
char *        strcat(char *string1, char *string2)
int           strcmp(char *string1, char *string2)
int           strncmp(char *string1, char *string2, unsigned long length)
char *        strcpy(char *to, char *from)
char *        strncpy(char *to, char *from, unsigned long length)
int           strlen(char *string)
```

Call a function from the main thread:

```
kern_return_t kern_serv_callout(kern_server_t *ksp, void (*func)(void *), void *arg)
```

## Network Functions

This section contains a summary of the network-specific kernel support functions, which are described in detail in Chapter 10. A general discussion of networking drivers and protocols is in Chapter 8, "Network Modules."

## Netif Functions

To use these functions, you need to include the header file **net/netif.h**.

Initialize and install a new netif:

```
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*)  
void **if\_register\_virtual**(if\_attach\_func\_t *attach\_func*, void \**private*)

Remove a netif:

void **if\_detach**(netif\_t *netif*)

Get or set data for a netif:

unsigned int **if\_collisions**(netif\_t *netif*)  
void **if\_collisions\_set**(netif\_t *netif*, unsigned int *collisions*)  
unsigned int **if\_flags**(netif\_t *netif*)  
void **if\_flags\_set**(netif\_t *netif*, unsigned int *flags*)  
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*)  
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*)  
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*)

Call a function implemented by a network module:

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*)

Get information about netifs:

netif\_class\_t **if\_class**(netif\_t *netif*)  
netif\_t **iflist\_first**(void)  
netif\_t **iflist\_next**(netif\_t *netif*)

Dispatch a packet to a protocol handler:

int **if\_handle\_input**(netif\_t *netif*, netbuf\_t *packet*, void \**extra*)

# Netbuf Functions

You should include the header file **net/netbuf.h** when you use these functions.

Allocate or free a netbuf or its wrapper:

netbuf\_t **nb\_alloc**(unsigned int *size*)  
netbuf\_t **nb\_alloc\_wrapper**(void \**data*, unsigned int *size*, void (\**freefunc*)(void \*), void \**freefunc\_arg*)  
void **nb\_free**(netbuf\_t *nb*)  
void **nb\_free\_wrapper**(netbuf\_t *nb*)

Change the size of a netbuf:

int **nb\_grow\_bot**(netbuf\_t *nb*, unsigned int *size*)  
int **nb\_shrink\_bot**(netbuf\_t *nb*, unsigned int *size*)

int	<b>nb_grow_top</b> (netbuf_t <i>nb</i> , unsigned int <i>size</i> )
int	<b>nb_shrink_top</b> (netbuf_t <i>nb</i> , unsigned int <i>size</i> )

Access the data in a netbuf:

char *	<b>nb_map</b> (netbuf_t <i>nb</i> )
int	<b>nb_read</b> (netbuf_t <i>nb</i> , unsigned int <i>offset</i> , unsigned int <i>size</i> , void * <i>target</i> )
int	<b>nb_write</b> (netbuf_t <i>nb</i> , unsigned int <i>offset</i> , unsigned int <i>size</i> , void * <i>source</i> )
unsigned int	<b>nb_size</b> (netbuf_t <i>nb</i> )

# Miscellaneous Functions

For the host-network conversion functions, you need to include the header file **netinet/in.h**. For **inet\_queue()**, you should include both **net/netif.h** and **net/netbuf.h**.

Convert values between host and network byte order:

u_long	<b>htonl</b> (u_long <i>hostlong</i> )
u_short	<b>htons</b> (u_short <i>hostshort</i> )
u_long	<b>ntohl</b> (u_long <i>netlong</i> )
u_short	<b>ntohs</b> (u_short <i>netshort</i> )

Give an IP input packet to the kernel for processing:

void	<b>inet_queue</b> (netif_t <i>netif</i> , netbuf_t <i>netbuf</i> )
------	--