Server Writer's Interface

Open Software Foundation and Carnegie Mellon University

Keith Loepere, Editor

OSF

May 6, 1991

This book is in the Open Software Foundation Mach 3 series.

Books in the OSF Mach 3 series:

MACH 3 Kernel Principles

MACH 3 Kernel Interface

Server Writer's Guide

Server Writer's Interface

Revision History:

Revision 0	November 3, 1990	Member's Meeting Version
Revision 0.1	May 6, 1991	Split Version

This document is partially derived from earlier Mach documents written by Eric Cooper, Richard P. Draves and Randall Dean.

Contents

CHAPTER 1 Introduction

This book documents the interfaces of use in writing a Mach server. The text describes each interface in isolation. The relationship of interfaces to one another, and the way that interfaces are combined to write user servers is the subject of the Server Writer's Guide.

Interface Descriptions

Each interface is listed separately, each starting on its own page. For each interface, some or all of the following features are presented:

- the name of the interface
- a brief description
- the pertinent library. For functions, this is the library that contains it and the header file that provides the function prototype. For data structures, this is the header file that defines it. If two such header files are listed, the first is the preferred file to use, and the file named in parenthesis is the actual file that declares the entity.
- a synopsis of the interface, in C form
- an extended description of the function performed by the call
- a description of each parameter to the call
- additional notes on the use of the interface
- cautions relating to the interface use
- an explanation of the significant return values
- references to related interfaces

General Notes

MIG Generated Interfaces

Some of the interfaces in this book are MIG generated interfaces. That is, they are stub routines generated from MIG interface description files. Calling these interfaces will actually result in a MACH IPC message being sent to the port that is the first argument in the call. This has two important effects.

- These calls may fail for various MIG or IPC related reasons. The list of error returns for these calls should always be considered to also include the IPC related errors (MACH_MSG_..., MACH_SEND_... and MACH_RCV_...) and the MIG related errors (MIG_...).
- These calls only invoke their expected effect when the acting port is indeed a port of the specified type. That is, if a call expects a port that names a task (a kernel task port) and the port is instead a port managed by a task, the routine will still happily generate the appropriate MACH message and send it to that task. What the target task will do with the message is up to it. Note that it is this effect that allows the Net message server to work.

Macros and Functions

Many of the interfaces described in this volume are actually macros. In general, the macros are written in such a way that they act exactly as if they were in-line functions. As such, they are documented just as if they were functions, except that they are given the type **Macro** instead o **Function**.

There are some exceptions to the macro as in-line function rule. The most prevalent is the macros that generate declarations. These macros are given the type **Declarative Macro**. They are to be used as documented in the same way as C declarations.

CHAPTER 2 C Thread Functions

This chapter describes functions that provide thread support for C programs.

Note that including **libthreads.a** redefines some system internal routines (**mig_init**, **mig_get_reply_port** and **mig_dealloc_reply_port**). **libthreads.a** must be linked prior to **libmach.a** when used.

condition_alloc

condition_alloc Macro — Dynamically allocate a condition variable

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

condition_t **condition_alloc**(*)*;

DESCRIPTION

The **condition_alloc** macro dynamically allocates and initializes a condition variable.

PARAMETERS

None

RETURN VALUE

A pointer to the condition variable.

RELATED INFORMATION

Functions: **condition_free**.

condition_broadcast

condition_broadcast Macro — Broadcast a status change in a condition variable

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **condition_broadcast**(condition_t *c*);

DESCRIPTION

The **condition_broadcast** macro indicates that a status change has occurred associated with condition variable *c*. All C threads waiting for this condition variable will be wakened.

PARAMETERS

c

A condition variable indicating the status change

RETURN VALUE

None

RELATED INFORMATION

Functions: **condition_signal**, **condition_wait**.

condition_clear

condition_clear Macro — Finalizes use of a user allocated condition variable

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **condition_clear**(condition_t *c*);

DESCRIPTION

The **condition_clear** macro finalizes use of a user allocated condition variable. In this context, a user allocated variable is one not obtained via **condition_alloc** (one initialized with **condition_init**). Finalizing a condition variable is also considered to broadcast the condition so associated.

PARAMETERS

c

A condition variable

RETURN VALUE

None

RELATED INFORMATION

Functions: **condition_init**.

condition_free

condition_free Macro — Free a dynamically allocated condition variable

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **condition_free**(condition_t *c)*;

DESCRIPTION

The **condition_free** macro frees a dynamically allocated condition variable (one obtained with **condition_alloc**). Freeing a condition variable is considered to broadcast the condition so associated.

PARAMETERS

c

A condition variable

RETURN VALUE

None

RELATED INFORMATION

Functions: **condition_alloc**.

condition_init

condition_init Macro — Initialize a user allocated condition variable

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **condition_init**(condition_t *c)*;

DESCRIPTION

The **condition_init** macro initializes a user allocated condition variable. In this context, a user allocated variable is one not obtained via **condition_alloc**.

PARAMETERS

c

A condition variable

RETURN VALUE

None

RELATED INFORMATION

Functions: **condition_clear**.

condition_name

condition_name Macro — Return a name associated with a condition variable

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

char ***condition_name**(condition_t *c)*;

DESCRIPTION

The **condition_name** macro returns the name associated with the given condition variable.IF there is no associated name, “?” is returned.

PARAMETERS

c

A condition variable

RETURN VALUE

A pointer to the associated name

RELATED INFORMATION

Functions: **condition_set_name**.

condition_set_name

condition_set_name Macro — Associate a name with a condition variable

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **condition_set_name**(condition_t *c*, char **name)*;

DESCRIPTION

The **condition_set_name** macro associates a name with a condition variable. Currently, these names are not used for anything; they can be retrieved with **condition_name**. Note that only a pointer to the name is associated with the condition variable; the name string must not be de-allocated until the name association is broken.

PARAMETERS

c

A condition variable

name

Name to associate

RETURN VALUE

None

RELATED INFORMATION

Functions: **condition_name**.

condition_signal

condition_signal Macro — Signal that a condition has occurred

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **condition_signal**(condition_t *c)*;

DESCRIPTION

The **condition_signal** macro indicates that a status change has occurred associated with condition variable *c*. At least one C thread waiting for this condition variable will be wakened.

PARAMETERS

c

A condition variable indicating the status change

RETURN VALUE

None.

RELATED INFORMATION

Functions: **condition_broadcast**, **condition_wait**.

condition_wait

condition_wait Function — Wait for a status change associated with a condition variable

LIBRARY

libthreads.a

SYNOPSIS

void **condition_wait**(condition_t *c*, mutex_t *m)*;

DESCRIPTION

The **condition_wait** function waits for a status change associated with some shared data. The calling thread is assumed to hold a mutex, *m*, protecting the data locked. This call releases the mutex and waits for the condition variable *c* to be signaled, indicating a change to the shared data. This call returns at some point in time after this event, with the mutex once again locked. The time between the signalling of the condition variable and the locking of the mutex is arbitrary; it is possible that some other thread could have locked the mutex and performed yet other changes (and condition signalling) prior to this thread re-obtaining the mutex.

PARAMETERS

c
: The condition variable indicating the status change

m
: A mutex that locks the data associated with the condition variable

NOTES

The typical use of this function is in a loop as follows:

```
mutex_t m;
condition_t c;
mutex_lock (m);
while (/* status of shared data is not okay */)
        condition_wait (c, m);
/* use shared data */
mutex_unlock (m);
```

RETURN VALUE

None.

RELATED INFORMATION

Functions: **condition_signal**, **condition_broadcast**.

cthread_count

cthread_count Function — Return the current number of C thread

LIBRARY

libthreads.a

SYNOPSIS

int **cthread_count**();

DESCRIPTION

The **cthread_count** function returns the current number of C threads. A C thread is counted as no longer existing when it returns from its top-level function (calls **cthread_exit**), not when some other thread detaches it or joins with it.

PARAMETERS

None

RETURN VALUE

Number of current C threads

RELATED INFORMATION

Functions: **cthread_fork**, **cthread_exit**.

cthread_data

cthread_data Macro — Returned data associated with a thread

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

any_t **cthread_data** (cthread_t *t)*;

DESCRIPTION

The **cthread_data** macro returns the data value associated with the given thread. This value provides a simple form of thread-specific “global” data. More elaborate mechanisms may be built upon this single value.

After a thread exits, any attempt to get or set its associated data is illegal, so any de-allocation or other cleanup of the data must be done before the thread exits. It is always safe to access the data associated with the caller’s own thread (**cthread_self**), or with a thread that has not yet been joined or detached.

PARAMETERS

t

A thread identifier

RETURN VALUE

The thread’s associated data value

RELATED INFORMATION

Functions: **cthread_set_data**.

cthread_detach

cthread_detach Function — Detach a C thread from all threads

LIBRARY

libthreads.a

SYNOPSIS

void **cthread_detach** (cthread_t *t)*;

DESCRIPTION

The **cthread_detach** function indicates that thread **t** will never be joined.

PARAMETERS

t

Thread identifier

NOTES

Since the fact that a thread is to be detached is normally known when it is created, this call is normally used as: **cthread_detach**(**cthread_fork**(*func*, *arg*));

RETURN VALUE

None.

RELATED INFORMATION

Functions: **cthread_fork**, **cthread_join**.

cthread_exit

cthread_exit Function — Terminate the current C thread

LIBRARY

libthreads.a

SYNOPSIS

void **cthread_exit** (any_t *result)*;

DESCRIPTION

The **cthread_exit** function terminates the calling thread. This call is implicit when the top-level function of a thread returns, in which case the argument to **cthread_exit** is the return value from the top-level function, but it can also be called explicitly. The *result* is made available to a thread that joins with this thread (**cthread_join**), or discarded if the thread is detached. If this is the first (main) thread, its termination will not terminate the task, but will instead wait for all other C threads to terminate and then terminate the task. The exit status for the task becomes the value of *result*.

PARAMETERS

result
: A value to be given to **cthread_join**

RETURN VALUE

None.

RELATED INFORMATION

Functions: **cthread_fork**, **cthread_join**, **cthread_detach**.

cthread_fork

cthread_fork Function — Create a new C thread

LIBRARY

libthreads.a

SYNOPSIS

cthread_t **cthread_fork** (any_t (**func*) (any_t *arg*), any_t *arg)*;

DESCRIPTION

The **cthread_fork** function creates a new thread which will execute concurrently with the current thread. This is the sole means of creating new threads. The new thread will execute the following call:

result = (**func*) (*arg*);

The result value from this call (assuming the call does not terminate itself via **cthread_exit**) is available via **cthread_join**. The call to **cthread_fork** returns a thread identifier useful for a call to **cthread_detach** or **cthread_join**. A thread may either be joined or detached only once. If the thread is neither joined nor detached, the thread's associated data will never be released.

PARAMETERS

func
: Top-level function to execute in the new thread.

arg
: Single argument to pass to *func*.

RETURN VALUE

A thread identifier naming the new thread.

RELATED INFORMATION

Functions: **cthread_exit**, **cthread_join**, **cthread_detach**.

cthread_init

cthread_init Function — Initialize the C threads package.

LIBRARY

libthreads.a

SYNOPSIS

int **cthread_init**();

DESCRIPTION

The **cthread_init** function initializes the C threads package. It is automatically called by **_start** when the C threads package is included when linking. This call also initializes the multi-threaded MIG routines. After this call, the initial point of control in the task becomes the first C thread. When this first thread terminates, it does not immediately terminate the task. It waits for all threads to terminate before so doing. The exit status for the task becomes the thread exit status of this first (main) thread.

PARAMETERS

None

RETURN VALUE

The top of the first thread's stack. (The return type of this function is incorrect.)

RELATED INFORMATION

Functions: **_start**.

cthread_join

cthread_join Function — Join with a C thread

LIBRARY

libthreads.a

SYNOPSIS

any_t **cthread_join** (cthread_t *t*);

DESCRIPTION

The **cthread_join** function suspends the caller until the thread *t* terminates.

PARAMETERS

t

A thread identifier

RETURN VALUE

Either the result of *t*'s top-level function (if it returned normally) or the argument with which *t* explicitly called **cthread_exit**

RELATED INFORMATION

Functions: **cthread_fork**, **cthread_detach**, **cthread_exit**.

cthread_kernel_limit

cthread_kernel_limit Function — Get the kernel thread limit for C threads

LIBRARY

libthreads.a

SYNOPSIS

int **cthread_kernel_limit**();

DESCRIPTION

The **cthread_kernel_limit** function returns the current limit on the number of kernel threads to use to support C threads. A value of zero is considered as no limit.

PARAMETERS

None

RETURN VALUE

The current kernel thread limit.

RELATED INFORMATION

Functions: **cthread_set_kernel_limit**, **cthread_limit**, **cthread_set_limit**.

cthread_limit

cthread_limit Function — Return the limit on active C threads

LIBRARY

libthreads.a

SYNOPSIS

int **cthread_limit**();

DESCRIPTION

The **cthread_limit** function returns the limit on the number of active C threads. In this context, a C threads is considered as active if it can be considered for execution by a supporting kernel thread (that is, it has an assigned *cproc*). The actual number of C threads that can actually be in execution at any time is governed by the thread kernel limit. A value of zero is considered as no limit.

PARAMETERS

None

RETURN VALUE

The number of allowed active C threads

RELATED INFORMATION

Functions: **cthread_set_limit**, **cthread_kernel_limit**, **cthread_set_kernel_-limit**.

cthread_mach_msg

cthread_mach_msg Function — C thread optimized MACH message routine

LIBRARY

libthreads.a

SYNOPSIS

mach_msg_return_t **cthread_mach_msg**(mach_msg_header_t *msg*,
mach_msg_option_t *option*, mach_msg_size_t
send_size, mach_msg_size_t *rcv_size*,
mach_port_t *rcv_name*, mach_msg_timeout_t
timeout, mach_port_t *notify*, int *min*, int *max)*;

DESCRIPTION

The **cthread_mach_msg** function performs a **mach_msg** call. (It is assumed that the MACH_RCV_MSG is specified.) This call differs from **mach_msg** only in as much as that this call limits the number of threads that may be actively servicing a given port (or port set). In this sense, "actively servicing" means that the C thread is allowed to wait for a message from the port. This call, as well as **cthread_msg_active**, declare a C thread to be actively servicing a port. When a C thread blocks performing a kernel function (such as **mach_msg**), it blocks its underlying MACH thread as well. If this thread's waiting would exceed the *max* value established at the first wait from the port, this thread will send its message (if there is one), but the C thread will block instead of doing the receive at this time. In this way, the underlying MACH thread is free to perform other work. When the number of C threads (and corresponding MACH threads) actively servicing this port falls below the *min* value, a C thread will be wakened to then perform its message receive (and become an active listener for this port as a result), blocking itself in the message receive (and thereby blocking its MACH thread as well).

A C thread cease to be an active receiver for a port when it calls **cthread_-mach_msg** with some different port, or when it calls **cthread_msg_busy**.

PARAMETERS

msg
: [in/out] A message buffer. This should be aligned on a long-word boundary.

option
: Refer to **mach_msg** for a description of this parameter.

send_size
: Refer to **mach_msg** for a description of this parameter.

rcv_size
Refer to **mach_msg** for a description of this parameter.

rcv_name
Refer to **mach_msg** for a description of this parameter.

timeout
Refer to **mach_msg** for a description of this parameter.

notify
Refer to **mach_msg** for a description of this parameter.

min
The maximum number of threads that can be left waiting for messages from *rcv_name* before other threads are released.

max
The maximum number of threads that can be waiting for messages from *rcv_name*.

RETURN VALUE

Return value from the **mach_msg** call.

RELATED INFORMATION

Functions: **mach_msg**, **cthread_msg_active**, **cthread_msg_busy**.

cthread_msg_active

cthread_msg_active Function — Mark this thread as actively servicing a port

LIBRARY

libthreads.a

SYNOPSIS

void **cthread_msg_active**(mach_port_t *port*, int *min*, int *max)*;

DESCRIPTION

The **cthread_msg_active** function declares that this C thread will be actively receiving (and thereby waiting for) messages from the specified port. By performing this call prior to any **cthread_mach_msg** calls for *port*, this thread is reserved as a listener for the port, and is guaranteed that it can receive without its C thread being blocked waiting for other threads to cease being active receivers for this port.

PARAMETERS

port

Receive port this thread will service

min

The maximum number of threads that can be left waiting for messages from *port* before other threads are released.

max

The maximum number of threads that can be waiting for messages from *port*.

RETURN VALUE

None

RELATED INFORMATION

Functions: **mach_msg**, **cthread_mach_msg**, **cthread_msg_busy**.

cthread_msg_busy

cthread_msg_busy Function — Cease to be an active receiver for a port

LIBRARY

libthreads.a

SYNOPSIS

void **cthread_msg_busy**(mach_port_t *port*, int *min*, int *max)*;

DESCRIPTION

The **cthread_msg_busy** function declares that this C thread will no longer be an active listener for its port. A thread is declared an active listener for a port either via **cthread_mach_msg** or **cthread_msg_active**. If, by releasing active listenership (either via this call or a **cthread_mach_msg** call specifying a different port), the active listeners falls below the minimum value for the port, a C thread will be wakened so it can perform its receive operation.

PARAMETERS

port
: Port for which this thread will no longer be a receiver

min
: Duplicates to match **cthread_msg_active**.

max
: Duplicates to match **cthread_msg_active**.

RETURN VALUE

None.

RELATED INFORMATION

Functions: **mach_msg**, **cthread_msg_active**, **cthread_mach_msg**.

cthread_name

cthread_name Function — Return the name associated with a thread

LIBRARY

libthreads.a

SYNOPSIS

char ***cthread_name** (cthread_t *t)*;

DESCRIPTION

The **cthread_name** function returns a pointer to the name associated with a thread. If the thread has no associated name, “?” is returned.

PARAMETERS

t

A thread identifier

RETURN VALUE

The thread’s associated name

RELATED INFORMATION

Functions: **cthread_set_name**.

cthread_self

cthread_self Macro — Return the caller's thread identifier

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

cthread_t **cthread_self**();

DESCRIPTION

The **cthread_self** macro returns the caller's own thread identifier, which is the same value that was returned by **cthread_fork** to the creator of the thread. The thread identifier uniquely identifies the thread, and hence may be used as a key in data structures that associate user data with individual threads. Since thread identifiers may be re-used by the underlying implementation, the programmer should be careful to clean up such associations when threads exit

PARAMETERS

None

RETURN VALUE

The thread's own identifier

RELATED INFORMATION

Functions: **cthread_fork**.

cthread_set_data

cthread_set_data Macro — Associate a data value with a thread

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **cthread_set_data**(cthread_t *t*, any_t *data)*;

DESCRIPTION

The **cthread_set_data** macro associates a single data value with a thread. This value may be subsequently retrieved by **cthread_data**.

PARAMETERS

t

A thread identifier

data

A single data value to associate with the thread

RETURN VALUE

None

RELATED INFORMATION

Functions: **cthread_data**.

cthread_set_kernel_limit

cthread_set_kernel_limit Function — Set the maximum number of kernel threads for C threads

LIBRARY

libthreads.a

SYNOPSIS

void **cthread_set_kernel_limit**(int *n)*;

DESCRIPTION

The **cthread_set_kernel_limit** function sets the limit on the number of kernel threads to use to support C threads. If the current number of kernel threads exceeds this value, none are destroyed. A value of zero is considered as no limit.

PARAMETERS

n

Maximum number of kernel threads

RETURN VALUE

None.

RELATED INFORMATION

Functions: **cthread_kernel_limit**, **cthread_limit**, **cthread_set_limit**.

cthread_set_limit

cthread_set_limit Function — Set the limit of active C threads

LIBRARY

libthreads.a

SYNOPSIS

void **cthread_set_limit** (int *n)*;

DESCRIPTION

The **cthread_set_limit** function limits the number of active C threads. In this context, a C thread is considered as active if it can be considered for execution by a supporting kernel thread (that is, it has an assigned *cproc*). The actual number of C threads that can actually be in execution at any time is governed by the thread kernel limit. A value of zero is considered as no limit.

PARAMETERS

n

Limit on active C threads

RETURN VALUE

None

RELATED INFORMATION

Functions: **cthread_limit**, **cthread_kernel_limit**, **cthread_set_kernel_limit**.

cthread_set_name

cthread_set_name Function — Associate a name with a thread

LIBRARY

libthreads.a

SYNOPSIS

void **cthread_set_name**(cthread_t *t*, char **name)*;

DESCRIPTION

The **cthread_set_name** function associates a name with a thread. Currently, these names are not used for anything; they can be retrieved with **cthread_name**. The initial thread is automatically given a name of "main". Note that only a pointer to the name is associated with the thread; the name string must not be de-allocated until the name association is broken.

PARAMETERS

t
: A thread identifier

name
: A name to associate

RETURN VALUE

None

RELATED INFORMATION

Functions: **cthread_name**.

cthread_sp

cthread_sp Function — Get the current stack pointer

LIBRARY

libthreads.a

SYNOPSIS

void ***cthread_sp**();

DESCRIPTION

The **cthread_sp** function returns the current stack pointer for the current C thread.

PARAMETERS

None

RETURN VALUE

The current stack pointer.

cthread_unwire

cthread_unwire Function — Un-bind a C thread from a MACH thread.

LIBRARY

libthreads.a

SYNOPSIS

void **cthread_unwire**(*)*;

DESCRIPTION

The **cthread_unwire** function breaks the binding of the current C thread from its MACH thread. After this call, the MACH thread is free to service any unbound C thread, and this C thread may be serviced by any unbound MACH thread.

PARAMETERS

None

RETURN VALUE

None.

RELATED INFORMATION

Functions: **cthread_wire**.

cthread_wire

cthread_wire Function — Bind a C thread to a MACH thread

LIBRARY

libthreads.a

SYNOPSIS

void **cthread_wire**();

DESCRIPTION

The **cthread_wire** function binds the calling C thread to the MACH thread currently executing it. After this, the current MACH thread is dedicated to running only this C thread, and this C thread will run using only this MACH thread. This is done to guarantee a free MACH thread for the activities of this C thread.

PARAMETERS

None

RETURN VALUE

None.

RELATED INFORMATION

Functions: **cthread_unwire**.

cthread_yield

cthread_yield Function — Schedule another thread

LIBRARY

libthreads.a

SYNOPSIS

void **cthread_yield**();

DESCRIPTION

The **cthread_yield** function provides a hint to the scheduler, suggesting that this would be a convenient point to schedule another thread to run on the current processor. If the current C thread is bound to a MACH thread, this call is equivalent to **switch_pri**. Otherwise, an attempt is made to use this MACH thread to service some other C thread; if no such runable C thread exists, **switch_pri** is called. Since multiple C threads can be serviced by a single MACH thread, and there is no pre-emption mechanism that will forcibly provide this servicing, this call may be needed to avoid starvation of threads.

PARAMETERS

None

RETURN VALUE

None.

RELATED INFORMATION

Functions: **swtch**, **swtch_pri**, **thread_switch**.

mutex_alloc

mutex_alloc Macro — Allocate a mutex variable

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

mutex_t **mutex_alloc**();

DESCRIPTION

The **mutex_alloc** macro allocates heap storage properly constructed as a mutex variable.

PARAMETERS

None

RETURN VALUE

A pointer to a mutex

RELATED INFORMATION

Functions: **mutex_free**.

mutex_clear

mutex_clear Macro — Finalize use of a user allocated mutex variable

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **mutex_clear** (mutex_t *m)*;

DESCRIPTION

The **mutex_clear** macro finalizes the use of a user allocated mutex variable. A user allocated mutex here means one for which the storage was obtained by the user in ways other than **mutex_alloc**, and subsequently initialized by **mutex_init**.

PARAMETERS

m

A mutex

RETURN VALUE

None

RELATED INFORMATION

Functions: **mutex_init**.

mutex_clear_special

mutex_clear_special Function — Mark a mutex non-“special”

LIBRARY

libthreads.a

SYNOPSIS

void **mutex_clear_special**(mutex_t *m)*;

DESCRIPTION

The **mutex_clear_special** function marks the mutex *m* as no longer “special”. A “special” mutex is one such that, when unlocked, waiting C threads will be run immediately.

PARAMETERS

m

A mutex

RETURN VALUE

None

RELATED INFORMATION

Functions: **mutex_set_special**.

mutex_free

mutex_free Macro — Free a dynamically allocated mutex variable

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **mutex_free** (mutex_t *m)*;

DESCRIPTION

The **mutex_free** macro frees a dynamically allocated mutex variable obtained via **mutex_alloc**.

PARAMETERS

m

A mutex

RETURN VALUE

None

RELATED INFORMATION

Functions: **mutex_alloc**.

mutex_init

mutex_init Macro — Initialize a user allocated mutex variable

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **mutex_init** (mutex_t *m*);

DESCRIPTION

The **mutex_init** macro initializes user allocated storage to be a mutex variable. In this context, user allocated storage is meant to be any storage other than that obtained via **mutex_alloc**.

PARAMETERS

m

A mutex variable

RETURN VALUE

None

RELATED INFORMATION

Functions: **mutex_clear**.

mutex_lock

mutex_lock Macro — Lock a mutex

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **mutex_lock** (mutex_t *m)*;

DESCRIPTION

The **mutex_lock** macro locks the specified mutex. It blocks until it succeeds. If several threads attempt to lock the same mutex concurrently, one will succeed, and the others will block until *m* is unlocked. The case of a thread attempting to lock a mutex it already holds is *not* treated specially; deadlock will result.

PARAMETERS

m

A mutex

RETURN VALUE

None

RELATED INFORMATION

Functions: **mutex_try_lock**, **mutex_unlock**.

mutex_name

mutex_name Macro — Return the name associated with a mutex

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

char ***mutex_name** (mutex_t *m)*;

DESCRIPTION

The **mutex_name** macro returns the name associated with a mutex. It the mutex has no associated name, “?” is returned.

PARAMETERS

m

A mutex

RETURN VALUE

The mutex’ associated name

RELATED INFORMATION

Functions: **mutex_set_name**.

mutex_set_name

mutex_set_name Macro — Associate a name with a mutex

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **mutex_set_name** (mutex_t *m*, char **name)*;

DESCRIPTION

The **mutex_set_name** macro associates a name with a mutex variable. Currently, these names are not used for anything; they can be retrieved with **mutex_name**. Note that only a pointer to the name is associated with the mutex variable; the name string must not be de-allocated until the name association is broken.

PARAMETERS

m
: A mutex

name
: Name to associate with *m*

RETURN VALUE

None

RELATED INFORMATION

Functions: **mutex_name**.

mutex_set_special

mutex_set_special Function — Mark a mutex as “special”

LIBRARY

libthreads.a

SYNOPSIS

void **mutex_set_special**(mutex_t *m)*;

DESCRIPTION

The **mutex_clear_special** function marks the mutex *m* as “special”. A “special” mutex is one such that, when unlocked, waiting C threads will be run immediately.

PARAMETERS

m

A mutex

RETURN VALUE

None

RELATED INFORMATION

Functions: **mutex_clear_special**.

mutex_try_lock

mutex_try_lock Macro — Attempt to lock a mutex

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

boolean_t **mutex_try_lock** (mutex_t *m*);

DESCRIPTION

The **mutex_try_lock** macro attempts to lock the mutex *m*, like **mutex_lock**. This macro does not block waiting for the mutex to become locked, returning a status in this case.

PARAMETERS

m
: A mutex

RETURN VALUE

TRUE
: The mutex is locked to this thread.

FALSE
: The mutex is locked to some other thread.

RELATED INFORMATION

Functions: **mutex_lock**, **mutex_unlock**.

mutex_unlock

mutex_unlock Macro — Unlock a mutex

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **mutex_unlock** (mutex_t *m)*;

DESCRIPTION

The **mutex_unlock** macro unlocks the specified mutex, giving other threads a chance to lock it.

PARAMETERS

m

A mutex

RETURN VALUE

None

RELATED INFORMATION

Functions: **mutex_lock**, **mutex_try_lock**.

spin_lock

spin_lock Macro — Lock a spin lock.

LIBRARY

libthreads.a

cthreads.h

SYNOPSIS

void **spin_lock** (int **p)*;

DESCRIPTION

The **spin_lock** macro locks the specified spin lock. It does not return until the lock is locked to this thread. A spin lock is a lower overhead lock than a mutex, and as a result lacks some of the functionality of a mutex. A spin lock is so named because a thread waiting for the lock "spins", wasting CPU time until the lock is released by the holding thread. (If the C threads package was built with the SPIN_RESCHED option, which it is by default, a **switch_pri** call will be done while waiting.) A spin lock is normally used to lock regions of short duration, when it is expected that any thread holding the lock will quickly release it.

PARAMETERS

p

The spin lock to lock.

RETURN VALUE

None.

RELATED INFORMATION

Functions: **spin_try_lock**, **spin_unlock**.

spin_try_lock

spin_try_lock Function — Attempt to lock a spin lock

LIBRARY

libthreads.a

SYNOPSIS

boolean_t **spin_try_lock** (int **p*);

DESCRIPTION

The **spin_try_lock** function. makes a single attempt to lock *p*. The call does not block if the attempt to lock is unsuccessful.

PARAMETERS

p
: The spin lock to lock.

RETURN VALUE

TRUE
: if the lock is now locked to this thread

FALSE
: if the lock is still locked to some other thread

RELATED INFORMATION

Functions: **spin_lock**, **spin_unlock**.

spin_unlock

spin_unlock Function — Unlock a spin lock

LIBRARY

libthreads.a

SYNOPSIS

void **spin_unlock** (int **p*);

DESCRIPTION

The **spin_unlock** function unlocks the specified spin lock. This routine does not check to see if the lock was locked, nor that it was locked to this thread.

PARAMETERS

p

The spin lock to be unlocked.

RETURN VALUE

None.

RELATED INFORMATION

Functions: **spin_try_lock**, **spin_lock**.

APPENDIX A C Language Functions

Not all of the functions and variables defined in ANSI C are available for use when writing a server. This appendix lists those items that do exist, and provides documentation for those that have differences or special features not found in a standard ANSI implementation.

The following functions exist, exactly as in ANSI/K&R C. They are all found in **libmach.a**:

bcopy	**bzero**	**environ**	**strcat**	**strcmp**
strcpy	**strlen**	**strncpy**		

The following functions exist in **libmach3.a**:

abs	**alloca**	**atoi**	**atol**	**bcmp**
bcopy	**bzero**	**index**	**memccpy**	**memchr**
memcmp	**memcpy**	**memset**	**snprintf**	**sprintf**
sscanf	**strcasecmp**	**strcat**	**strcmp**	**strcpy**
strlen	**strncasecmp**	**strncat**	**strncmp**	**strncpy**
swab	**vsprintf**	**vsscanf**		

The following area related functions exist. They function just like the ANSI equivalents. They have the additional property of functioning correctly in the presence of multiple C threads. They exist in **libthreads.a**.

free	**malloc**	**realloc**

In addition, the following related functions (which use the above functions) appear in **libmach3.a**:

calloc	**cfree**

The following C functions differ from their normal counterparts as follows:

_start Function

The **_start** function performs C run-time, C thread, MIG and MACH related task start-up functions. This call occurs automatically when a task starts.

exit Function

The **exit** function terminates the calling task. It is equivalent to **task_terminate** (**task_self**()).

longjmp Function

The **longjmp** routine is useful for dealing with errors and interrupts encountered in a low-level subroutine of a program. **longjmp** restores the environment saved by a previous call to **setjmp**. The **longjmp** call returns in such a way that execution continues as if the call of **setjmp** had just returned the value supplied to **longjmp** to the function that invoked **setjmp**, which must not itself have returned in the interim. All accessible data have values as of the time **longjmp** was called.

The **libmach longjmp** function differs from its C counterpart in that it does not manipulate signal mask state.

setjmp Function

The **setjmp** routine is useful for dealing with errors and interrupts encountered in a low-level subroutine of a program. When called, it saves its stack environment in *env*. At this time, it returns 0.

A subsequent **longjmp** restores the environment so saved. The **longjmp** call then returns in such a way that execution continues as if the call of **setjmp** had just returned the value supplied to **longjmp** to the function that invoked **setjmp**, which must not itself have returned in the interim. All accessible data have values as of the time **longjmp** was called.

The **libmach setjmp** function differs from its C counterpart in that it does not manipulate signal mask state.

APPENDIX B

Function and Data Structure Index