

**NAME**

**call\_once, cnd\_broadcast, cnd\_destroy, cnd\_init, cnd\_signal, cnd\_timedwait, cnd\_wait, mtx\_destroy, mtx\_init, mtx\_lock, mtx\_timedlock, mtx\_trylock, mtx\_unlock, thrд\_create, thrд\_current, thrд\_detach, thrд\_equal, thrд\_exit, thrд\_join, thrд\_sleep, thrд\_yield, tss\_create, tss\_delete, tss\_get, tss\_set** - C11 threads interface

**LIBRARY**

C11 Threads Library (libstdthreads, -lstdthreads)

**SYNOPSIS**

```
#include <threads.h>
```

*void*

```
call_once(once_flag *flag, void (*func)(void));
```

*int*

```
cnd_broadcast(cnd_t *cnd);
```

*void*

```
cnd_destroy(cnd_t *cnd);
```

*int*

```
cnd_init(cnd_t *cnd);
```

*int*

```
cnd_signal(cnd_t *cnd);
```

*int*

```
cnd_timedwait(cnd_t * restrict cnd, mtx_t * restrict mtx, const struct timespec * restrict ts);
```

*int*

```
cnd_wait(cnd_t *cnd, mtx_t *mtx);
```

*void*

```
mtx_destroy(mtx_t *mtx);
```

*int*

```
mtx_init(mtx_t *mtx, int type);
```

*int*

**mtx\_lock**(*mtx\_t* \**mtx*);

*int*

**mtx\_timedlock**(*mtx\_t* \* *restrict* *mtx*, *const struct timespec* \* *restrict* *ts*);

*int*

**mtx\_trylock**(*mtx\_t* \**mtx*);

*int*

**mtx\_unlock**(*mtx\_t* \**mtx*);

*int*

**thrd\_create**(*thrd\_t* \**thr*, *int* (\**func*)(*void* \*), *void* \**arg*);

*thrd\_t*

**thrd\_current**(*void*);

*int*

**thrd\_detach**(*thrd\_t* *thr*);

*int*

**thrd\_equal**(*thrd\_t* *thr0*, *thrd\_t* *thr1*);

*\_Noreturn void*

**thrd\_exit**(*int* *res*);

*int*

**thrd\_join**(*thrd\_t* *thr*, *int* \**res*);

*int*

**thrd\_sleep**(*const struct timespec* \**duration*, *struct timespec* \**remaining*);

*void*

**thrd\_yield**(*void*);

*int*

**tss\_create**(*tss\_t* \**key*, *void* (\**dtor*)(*void* \*));

*void*

**tss\_delete**(*tss\_t* *key*);

*void \**

**tss\_get**(*tss\_t key*);

*int*

**tss\_set**(*tss\_t key, void \*val*);

## DESCRIPTION

As of ISO/IEC 9899:2011 ("ISO C11"), the C standard includes an API for writing multithreaded applications. Since POSIX.1 already includes a threading API that is used by virtually any multithreaded application, the interface provided by the C standard can be considered superfluous.

In this implementation, the threading interface is therefore implemented as a light-weight layer on top of existing interfaces. The functions to which these routines are mapped, are listed in the following table.

Please refer to the documentation of the POSIX equivalent functions for more information.

<i>Function</i>	<i>POSIX equivalent</i>
<b>call_once</b> ()	pthread_once(3)
<b>cond_broadcast</b> ()	pthread_cond_broadcast(3)
<b>cond_destroy</b> ()	pthread_cond_destroy(3)
<b>cond_init</b> ()	pthread_cond_init(3)
<b>cond_signal</b> ()	pthread_cond_signal(3)
<b>cond_timedwait</b> ()	pthread_cond_timedwait(3)
<b>cond_wait</b> ()	pthread_cond_wait(3)
<b>mtx_destroy</b> ()	pthread_mutex_destroy(3)
<b>mtx_init</b> ()	pthread_mutex_init(3)
<b>mtx_lock</b> ()	pthread_mutex_lock(3)
<b>mtx_timedlock</b> ()	pthread_mutex_timedlock(3)
<b>mtx_trylock</b> ()	pthread_mutex_trylock(3)
<b>mtx_unlock</b> ()	pthread_mutex_unlock(3)
<b>thrd_create</b> ()	pthread_create(3)
<b>thrd_current</b> ()	pthread_self(3)
<b>thrd_detach</b> ()	pthread_detach(3)
<b>thrd_equal</b> ()	pthread_equal(3)
<b>thrd_exit</b> ()	pthread_exit(3)
<b>thrd_join</b> ()	pthread_join(3)
<b>thrd_sleep</b> ()	nanosleep(2)
<b>thrd_yield</b> ()	pthread_yield(3)
<b>tss_create</b> ()	pthread_key_create(3)
<b>tss_delete</b> ()	pthread_key_delete(3)
<b>tss_get</b> ()	pthread_getspecific(3)

`tss_set()``pthread_setspecific(3)`

## DIFFERENCES WITH POSIX EQUIVALENTS

The `thrd_exit()` function returns an integer value to the thread calling `thrd_join()`, whereas the `pthread_exit()` function uses a pointer.

The mutex created by `mtx_init()` can be of *type* `mtx_plain` or `mtx_timed` to distinguish between a mutex that supports `mtx_timedlock()`. This type can be *or'd* with `mtx_recursive` to create a mutex that allows recursive acquisition. These properties are normally set using `pthread_mutex_init()`'s *attr* parameter.

## RETURN VALUES

If successful, the `cnd_broadcast()`, `cnd_init()`, `cnd_signal()`, `cnd_timedwait()`, `cnd_wait()`, `mtx_init()`, `mtx_lock()`, `mtx_timedlock()`, `mtx_trylock()`, `mtx_unlock()`, `thrd_create()`, `thrd_detach()`, `thrd_equal()`, `thrd_join()`, `thrd_sleep()`, `tss_create()` and `tss_set()` functions return `thrd_success`. Otherwise an error code will be returned to indicate the error.

The `thrd_current()` function returns the thread ID of the calling thread.

The `tss_get()` function returns the thread-specific data value associated with the given *key*. If no thread-specific data value is associated with *key*, then the value `NULL` is returned.

## ERRORS

The `cnd_init()` and `thrd_create()` functions will fail if:

`thrd_nomem` The system has insufficient memory.

The `cnd_timedwait()` and `mtx_timedlock()` functions will fail if:

`thrd_timedout` The system time has reached or exceeded the time specified in *ts* before the operation could be completed.

The `mtx_trylock()` function will fail if:

`thrd_busy` The mutex is already locked.

In all other cases, these functions may fail by returning general error code `thrd_error`.

## SEE ALSO

`nanosleep(2)`, `pthread(3)`

**STANDARDS**

These functions are expected to conform to ISO/IEC 9899:2011 ("ISO C11").

**HISTORY**

These functions appeared in FreeBSD 10.0.

**AUTHORS**

Ed Schouten <*ed@FreeBSD.org*>