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**, **thrd_create**, **thrd_current**, **thrd_detach**, **thrd_equal**, **thrd_exit**, **thrd_join**, **thrd_sleep**, **thrd_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 *cond*);

void
cnd_destroy(*cnd_t *cond*);

int
cnd_init(*cnd_t *cond*);

int
cnd_signal(*cnd_t *cond*);

int
cnd_timedwait(*cnd_t * restrict cond, mtx_t * restrict mtx, const struct timespec * restrict ts*);

int
cnd_wait(*cnd_t *cond, 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.

Function	*POSIX equivalent*
call_once()	pthread_once(3)
cnd_broadcast()	pthread_cond_broadcast(3)
cnd_destroy()	pthread_cond_destroy(3)
cnd_init()	pthread_cond_init(3)
cnd_signal()	pthread_cond_signal(3)
cnd_timedwait()	pthread_cond_timedwait(3)
cnd_wait()	pthread_cond_wait(3)
mtx_destroy()	pthread_mutex_destroy(3)
mtx_init()	pthread_mutex_init(3)
mtx_lock()	pthread_mutex_lock(3)
mtx_timedlock()	pthread_mutex_timedlock(3)
mtx_trylock()	pthread_mutex_trylock(3)
mtx_unlock()	pthread_mutex_unlock(3)
thrd_create()	pthread_create(3)
thrd_current()	pthread_self(3)
thrd_detach()	pthread_detach(3)
thrd_equal()	pthread_equal(3)
thrd_exit()	pthread_exit(3)
thrd_join()	pthread_join(3)
thrd_sleep()	nanosleep(2)
thrd_yield()	pthread_yield(3)
tss_create()	pthread_key_create(3)
tss_delete()	pthread_key_delete(3)
tss_get()	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*>