Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'pthread_cond_destroy.3p' command

$ man pthread_cond_destroy.3p

PTHREAD_COND_DESTROY(3P) POSIX Programmer's Manual PTHREAD_COND_DESTROY(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_cond_destroy, pthread_cond_init ? destroy and initialize condi? tion variables

SYNOPSIS

```
#include <pthread.h>
int pthread_cond_destroy(pthread_cond_t *cond);
int pthread_cond_init(pthread_cond_t *restrict cond,
    const pthread_condattr_t *restrict attr);
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

DESCRIPTION

The pthread_cond_destroy() function shall destroy the given condition variable specified by cond; the object becomes, in effect, uninitial? ized. An implementation may cause pthread_cond_destroy() to set the ob? ject referenced by cond to an invalid value. A destroyed condition variable object can be reinitialized using pthread_cond_init(); the re? sults of otherwise referencing the object after it has been destroyed are undefined.

It shall be safe to destroy an initialized condition variable upon which no threads are currently blocked. Attempting to destroy a condi? tion variable upon which other threads are currently blocked results in undefined behavior.

The pthread_cond_init() function shall initialize the condition vari? able referenced by cond with attributes referenced by attr. If attr is NULL, the default condition variable attributes shall be used; the ef? fect is the same as passing the address of a default condition variable attributes object. Upon successful initialization, the state of the condition variable shall become initialized.

See Section 2.9.9, Synchronization Object Copies and Alternative Map? pings for further requirements.

Attempting to initialize an already initialized condition variable re? sults in undefined behavior.

In cases where default condition variable attributes are appropriate, the macro PTHREAD_COND_INITIALIZER can be used to initialize condition variables. The effect shall be equivalent to dynamic initialization by a call to pthread_cond_init() with parameter attr specified as NULL, except that no error checks are performed.

The behavior is undefined if the value specified by the cond argument to pthread_cond_destroy() does not refer to an initialized condition variable.

The behavior is undefined if the value specified by the attr argument to pthread_cond_init() does not refer to an initialized condition vari? able attributes object.

RETURN VALUE

If successful, the pthread_cond_destroy() and pthread_cond_init() func? tions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The pthread_cond_init() function shall fail if:

EAGAIN The system lacked the necessary resources (other than memory) to initialize another condition variable.

ENOMEM Insufficient memory exists to initialize the condition variable.

These functions shall not return an error code of [EINTR].

The following sections are informative.

EXAMPLES

A condition variable can be destroyed immediately after all the threads that are blocked on it are awakened. For example, consider the follow? ing code:

```
struct list {
    pthread_mutex_t lm;
    ...
}
struct elt {
    key k;
    int busy;
    pthread_cond_t notbusy;
    ...
}
/* Find a list element and reserve it. */
struct elt *
list_find(struct list *lp, key k)
{
    struct elt *ep;
    pthread_mutex_lock(&lp->lm);
    while ((ep = find_elt(l, k) != NULL) && ep->busy)
        pthread_cond_wait(&ep->notbusy, &lp->lm);
    if (ep != NULL)
        ep->busy = 1;
    pthread_mutex_unlock(&lp->lm);
    return(ep);
}
delete_elt(struct list *lp, struct elt *ep)
{
    pthread_mutex_lock(&lp->lm);
```

```
    assert(ep->busy);
    ... remove ep from list ...
    ep->busy = 0;  /* Paranoid. */
(A) pthread_cond_broadcast(&ep->notbusy);
    pthread_mutex_unlock(&lp->lm);
(B) pthread_cond_destroy(&ep->notbusy);
    free(ep);
}
```

In this example, the condition variable and its list element may be freed (line B) immediately after all threads waiting for it are awak? ened (line A), since the mutex and the code ensure that no other thread can touch the element to be deleted.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the cond argu? ment to pthread_cond_destroy() does not refer to an initialized condi? tion variable, it is recommended that the function should fail and re? port an [EINVAL] error.

If an implementation detects that the value specified by the cond argu? ment to pthread_cond_destroy() or pthread_cond_init() refers to a con? dition variable that is in use (for example, in a pthread_cond_wait() call) by another thread, or detects that the value specified by the cond argument to pthread_cond_init() refers to an already initialized condition variable, it is recommended that the function should fail and report an [EBUSY] error.

If an implementation detects that the value specified by the attr argu? ment to pthread_cond_init() does not refer to an initialized condition variable attributes object, it is recommended that the function should fail and report an [EINVAL] error.

See also pthread_mutex_destroy().

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_broadcast(), pthread_cond_timedwait(), pthread_mutex_de?
stroy()

The Base Definitions volume of POSIX.1?2017, <pthread.h>

COPYRIGHT