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_key_delete.3p' command

$ man pthread_key_delete.3p

PTHREAD_KEY_DELETE(3P) POSIX Programmer's Manual PTHREAD_KEY_DELETE(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_key_delete ? thread-specific data key deletion

SYNOPSIS

#include <pthread.h>

int pthread_key_delete(pthread_key_t key);

DESCRIPTION

The pthread_key_delete() function shall delete a thread-specific data key previously returned by pthread_key_create(). The thread-specific data values associated with key need not be NULL at the time pthread_key_delete() is called. It is the responsibility of the appli? cation to free any application storage or perform any cleanup actions for data structures related to the deleted key or associated thread-specific data in any threads; this cleanup can be done either before or after pthread_key_delete() is called. Any attempt to use key following the call to pthread_key_delete() results in undefined behavior.

The pthread_key_delete() function shall be callable from within de? structor functions. No destructor functions shall be invoked by

pthread_key_delete(). Any destructor function that may have been asso?
ciated with key shall no longer be called upon thread exit.

RETURN VALUE

If successful, the pthread_key_delete() function shall return zero;
otherwise, an error number shall be returned to indicate the error.

ERRORS

The pthread_key_delete() function shall not return an error code of
[EINTR].

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

A thread-specific data key deletion function has been included in order
to allow the resources associated with an unused thread-specific data
key to be freed. Unused thread-specific data keys can arise, among
other scenarios, when a dynamically loaded module that allocated a key
is unloaded.

Conforming applications are responsible for performing any cleanup ac?
tions needed for data structures associated with the key to be deleted,
including data referenced by thread-specific data values. No such
cleanup is done by pthread_key_delete(). In particular, destructor
functions are not called. There are several reasons for this division
of responsibility:

1. The associated destructor functions used to free thread-specific
   data at thread exit time are only guaranteed to work correctly when
   called in the thread that allocated the thread-specific data. (De?
   structors themselves may utilize thread-specific data.) Thus, they
   cannot be used to free thread-specific data in other threads at key
   deletion time. Attempting to have them called by other threads at
   key deletion time would require other threads to be asynchronously
   interrupted. But since interrupted threads could be in an arbitrary

state, including holding locks necessary for the destructor to run, this approach would fail. In general, there is no safe mechanism whereby an implementation could free thread-specific data at key deletion time.

2. Even if there were a means of safely freeing thread-specific data associated with keys to be deleted, doing so would require that im? plementations be able to enumerate the threads with non-NULL data and potentially keep them from creating more thread-specific data while the key deletion is occurring. This special case could cause extra synchronization in the normal case, which would otherwise be unnecessary.

For an application to know that it is safe to delete a key, it has to know that all the threads that might potentially ever use the key do not attempt to use it again. For example, it could know this if all the client threads have called a cleanup procedure declaring that they are through with the module that is being shut down, perhaps by setting a reference count to zero.

If an implementation detects that the value specified by the key argu? ment to pthread_key_delete() does not refer to a a key value obtained from pthread_key_create() or refers to a key that has been deleted with pthread_key_delete(), it is recommended that the function should fail and report an [EINVAL] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_key_create()

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

COPYRIGHT

event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .