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_join.3p' command

$ man pthread_join.3p

PTHREAD_JOIN(3P) POSIX Programmer's Manual PTHREAD_JOIN(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_join ? wait for thread termination

SYNOPSIS

#include <pthread.h>

int pthread_join(pthread_t thread, void **value_ptr);

DESCRIPTION

The pthread_join() function shall suspend execution of the calling thread until the target thread terminates, unless the target thread has already terminated. On return from a successful pthread_join() call with a non-NULL value_ptr argument, the value passed to pthread_exit() by the terminating thread shall be made available in the location ref? erenced by value_ptr. When a pthread_join() returns successfully, the target thread has been terminated. The results of multiple simultaneous calls to pthread_join() specifying the same target thread are unde? fined. If the thread calling pthread_join() is canceled, then the tar? get thread shall not be detached.

It is unspecified whether a thread that has exited but remains unjoined

counts against {PTHREAD_THREADS_MAX}.

The behavior is undefined if the value specified by the thread argument to pthread_join() does not refer to a joinable thread.

The behavior is undefined if the value specified by the thread argument to pthread_join() refers to the calling thread.

RETURN VALUE

If successful, the pthread_join() function shall return zero; other?wise, an error number shall be returned to indicate the error.

ERRORS

The pthread_join() function may fail if:

EDEADLK

A deadlock was detected.

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

The following sections are informative.

EXAMPLES

An example of thread creation and deletion follows:

```
typedef struct {
    int *ar;
    long n;
} subarray;
void *
incer(void *arg)
{
    long i;
    for (i = 0; i < ((subarray *)arg)->n; i++)
        ((subarray *)arg)->ar[i]++;
}
int main(void)
{
    int        ar[1000000];
    pthread_t  th1, th2;
    subarray   sb1, sb2;
    sb1.ar = &ar[0];
```

```
        sb1.n  = 500000;
        (void) pthread_create(&th1, NULL, incer, &sb1);
        sb2.ar = &ar[500000];
        sb2.n  = 500000;
        (void) pthread_create(&th2, NULL, incer, &sb2);
        (void) pthread_join(th1, NULL);
        (void) pthread_join(th2, NULL);
        return 0;
    }
```

APPLICATION USAGE

None.

RATIONALE

The pthread_join() function is a convenience that has proven useful in multi-threaded applications. It is true that a programmer could simu? late this function if it were not provided by passing extra state as part of the argument to the start_routine(). The terminating thread would set a flag to indicate termination and broadcast a condition that is part of that state; a joining thread would wait on that condition variable. While such a technique would allow a thread to wait on more complex conditions (for example, waiting for multiple threads to termi? nate), waiting on individual thread termination is considered widely useful. Also, including the pthread_join() function in no way precludes a programmer from coding such complex waits. Thus, while not a primi? tive, including pthread_join() in this volume of POSIX.1?2017 was con? sidered valuable.

The pthread_join() function provides a simple mechanism allowing an ap? plication to wait for a thread to terminate. After the thread termi? nates, the application may then choose to clean up resources that were used by the thread. For instance, after pthread_join() returns, any ap? plication-provided stack storage could be reclaimed.

The pthread_join() or pthread_detach() function should eventually be called for every thread that is created with the detachstate attribute set to PTHREAD_CREATE_JOINABLE so that storage associated with the

thread may be reclaimed.

The interaction between pthread_join() and cancellation is well-defined for the following reasons:

* The pthread_join() function, like all other non-async-cancel-safe functions, can only be called with deferred cancelability type.

* Cancellation cannot occur in the disabled cancelability state.

Thus, only the default cancelability state need be considered. As specified, either the pthread_join() call is canceled, or it succeeds, but not both. The difference is obvious to the application, since either a cancellation handler is run or pthread_join() returns. There are no race conditions since pthread_join() was called in the deferred cancelability state.

If an implementation detects that the value specified by the thread argument to pthread_join() does not refer to a joinable thread, it is recommended that the function should fail and report an [EINVAL] error.

If an implementation detects that the value specified by the thread argument to pthread_join() refers to the calling thread, it is recommended that the function should fail and report an [EDEADLK] error.

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an [ESRCH] error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create(), wait()

The Base Definitions volume of POSIX.1?2017, Section 4.12, Memory Synchronization, <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 .