



Linux Ubuntu 26.04 Manual Pages on command 'pthread_attr_init.3'

\$ man pthread_attr_init.3

pthread_attr_init(3) Library Functions Manual pthread_attr_init(3)

NAME

pthread_attr_init, pthread_attr_destroy - initialize and destroy thread attributes object

LIBRARY

POSIX threads library (libpthread, -lpthread)

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_init(pthread_attr_t *attr);

int pthread_attr_destroy(pthread_attr_t *attr);
```

DESCRIPTION

The pthread_attr_init() function initializes the thread attributes object pointed to by attr with default attribute values. After this call, individual attributes of the object can be set using various related functions (listed under SEE ALSO), and then the object can be used in one or more pthread_create(3) calls that create threads.

Calling pthread_attr_init() on a thread attributes object that has already been initialized results in undefined behavior.

When a thread attributes object is no longer required, it should be destroyed using the pthread_attr_destroy() function. Destroying a thread attributes object has no effect on threads that were created using that object.

Once a thread attributes object has been destroyed, it can be reinitialized using pthread_attr_init(). Any other use of a destroyed thread attributes object has undefined results.

RETURN VALUE

ERRORS

ATTRIBUTES

[illegible][illegible][illegible]

POSIX.1-2008.

POSIX.1-2001.

The `pthread_attr_t` type should be treated as opaque: any access to the object other than via pthreads functions is nonportable and produces undefined results.

The program below optionally makes use of `pthread_attr_init()` and various related functions to initialize a thread attributes object that is used to create a single thread. Once created, the thread uses the `pthread_getattr_np(3)` function (a nonstandard GNU extension) to retrieve the thread's attributes, and then displays those attributes.

If the program is run with no command-line argument, then it passes NULL as the `attr` argument of `pthread_create(3)`, so that the thread is created with default attributes. Running the program on Linux/x86-32 with the NPTL threading implementation, we see the following:

unlimited

Thread attributes:

Scope = PTHREAD_SCOPE_SYSTEM

Inherit scheduler = PTHREAD_INHERIT_SCHED

Scheduling policy = SCHED_OTHER

Scheduling priority = 0

Guard size = 4096 bytes

Stack address = 0x40196000

Stack size = 0x201000 bytes

When we supply a stack size as a command-line argument, the program initializes a thread at?

tributes object, sets various attributes in that object, and passes a pointer to the object

in the call to pthread_create(3). Running the program on Linux/x86-32 with the NPTL thread?

ing implementation, we see the following:

```
$ ./a.out 0x3000000
```

posix_memalign() allocated at 0x40197000

Thread attributes:

Detach state = PTHREAD_CREATE_DETACHED

Scope = PTHREAD_SCOPE_SYSTEM

Inherit scheduler = PTHREAD_EXPLICIT_SCHED

Scheduling policy = SCHED_OTHER

Scheduling priority = 0

Guard size = 0 bytes

Stack address = 0x40197000

Stack size = 0x3000000 bytes

Program source

```
#define _GNU_SOURCE /* To get pthread_getattr_np() declaration */
```

```
#include <err.h>
```

```
#include <errno.h>
```

```
#include <pthread.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
static void
```

```
display_pthread_attr(pthread_attr_t *attr, char *prefix)
```

```
{
```

```
    int s, i;
```

```

size_t v;

void *stkaddr;

struct sched_param sp;

s = pthread_attr_getdetachstate(attr, &i);

if (s != 0)

    errc(EXIT_FAILURE, s, "pthread_attr_getdetachstate");

printf("%sDetach state      = %s\n", prefix,

        (i == PTHREAD_CREATE_DETACHED) ? "PTHREAD_CREATE_DETACHED" :

        (i == PTHREAD_CREATE_JOINABLE) ? "PTHREAD_CREATE_JOINABLE" :

        "???");

s = pthread_attr_getscope(attr, &i);

if (s != 0)

    errc(EXIT_FAILURE, s, "pthread_attr_getscope");

printf("%sScope              = %s\n", prefix,

        (i == PTHREAD_SCOPE_SYSTEM) ? "PTHREAD_SCOPE_SYSTEM" :

        (i == PTHREAD_SCOPE_PROCESS) ? "PTHREAD_SCOPE_PROCESS" :

        "???");

s = pthread_attr_getinheritsched(attr, &i);

if (s != 0)

    errc(EXIT_FAILURE, s, "pthread_attr_getinheritsched");

printf("%sInherit scheduler   = %s\n", prefix,

        (i == PTHREAD_INHERIT_SCHED) ? "PTHREAD_INHERIT_SCHED" :

        (i == PTHREAD_EXPLICIT_SCHED) ? "PTHREAD_EXPLICIT_SCHED" :

        "???");

s = pthread_attr_getschedpolicy(attr, &i);

if (s != 0)

    errc(EXIT_FAILURE, s, "pthread_attr_getschedpolicy");

printf("%sScheduling policy    = %s\n", prefix,

        (i == SCHED_OTHER) ? "SCHED_OTHER" :

        (i == SCHED_FIFO) ? "SCHED_FIFO" :

        (i == SCHED_RR) ? "SCHED_RR" :

        "???");

s = pthread_attr_getschedparam(attr, &sp);

```

```

if (s != 0)
    errc(EXIT_FAILURE, s, "pthread_attr_getschedparam");
printf("%sScheduling priority = %d\n", prefix, sp.sched_priority);
s = pthread_attr_getguardsize(attr, &v);
if (s != 0)
    errc(EXIT_FAILURE, s, "pthread_attr_getguardsize");
printf("%sGuard size      = %zu bytes\n", prefix, v);
s = pthread_attr_getstack(attr, &stkaddr, &v);
if (s != 0)
    errc(EXIT_FAILURE, s, "pthread_attr_getstack");
printf("%sStack address    = %p\n", prefix, stkaddr);
printf("%sStack size      = %#zx bytes\n", prefix, v);
}

static void *
thread_start(void *arg)
{
    int s;
    pthread_attr_t gattr;

    /* pthread_getattr_np() is a non-standard GNU extension that
       retrieves the attributes of the thread specified in its
       first argument. */
    s = pthread_getattr_np(pthread_self(), &gattr);
    if (s != 0)
        errc(EXIT_FAILURE, s, "pthread_getattr_np");
    printf("Thread attributes:\n");
    display_pthread_attr(&gattr, "\t");
    exit(EXIT_SUCCESS);    /* Terminate all threads */
}

int
main(int argc, char *argv[])
{
    pthread_t thr;
    pthread_attr_t attr;

```

```

pthread_attr_t *attrp;    /* NULL or &attr */

int s;

attrp = NULL;

/* If a command-line argument was supplied, use it to set the
   stack-size attribute and set a few other thread attributes,
   and set attrp pointing to thread attributes object. */
if (argc > 1) {
    size_t stack_size;

    void *sp;

    attrp = &attr;

    s = pthread_attr_init(&attr);

    if (s != 0)
        errc(EXIT_FAILURE, s, "pthread_attr_init");

    s = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

    if (s != 0)
        errc(EXIT_FAILURE, s, "pthread_attr_setdetachstate");

    s = pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);

    if (s != 0)
        errc(EXIT_FAILURE, s, "pthread_attr_setinheritsched");

    stack_size = strtoul(argv[1], NULL, 0);

    s = posix_memalign(&sp, sysconf(_SC_PAGESIZE), stack_size);

    if (s != 0)
        errc(EXIT_FAILURE, s, "posix_memalign");

    printf("posix_memalign() allocated at %p\n", sp);

    s = pthread_attr_setstack(&attr, sp, stack_size);

    if (s != 0)
        errc(EXIT_FAILURE, s, "pthread_attr_setstack");
}

s = pthread_create(&thr, attrp, &thread_start, NULL);

if (s != 0)
    errc(EXIT_FAILURE, s, "pthread_create");

if (attrp != NULL) {
    s = pthread_attr_destroy(attrp);

```

```
    if (s != 0)
        errc(EXIT_FAILURE, s, "pthread_attr_destroy");
}

pause(); /* Terminates when other thread calls exit() */
}
```

SEE ALSO

pthread_attr_setaffinity_np(3), pthread_attr_setdetachstate(3),
pthread_attr_setguardsize(3), pthread_attr_setinheritsched(3),
pthread_attr_setschedparam(3), pthread_attr_setschedpolicy(3), pthread_attr_setscope(3),
pthread_attr_setsigmask_np(3), pthread_attr_setstack(3), pthread_attr_setstackaddr(3),
pthread_attr_setstacksize(3), pthread_create(3), pthread_getattr_np(3),
pthread_setattr_default_np(3), pthreads(7)

Linux man-pages 6.17

2025-09-21

pthread_attr_init(3)