



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

\$ man pthread_getattr_np.3

pthread_getattr_np(3) Library Functions Manual pthread_getattr_np(3)

NAME

pthread_getattr_np - get attributes of created thread

LIBRARY

POSIX threads library (libpthread, -lpthread)

SYNOPSIS

```
#define _GNU_SOURCE                      /* See feature_test_macros(7) */  
  
#include <pthread.h>  
  
int pthread_getattr_np(pthread_t thread, pthread_attr_t *attr);
```

DESCRIPTION

The pthread_getattr_np() function initializes the thread attributes object referred to by attr so that it contains actual attribute values describing the running thread thread.

The returned attribute values may differ from the corresponding attribute values passed in the attr object that was used to create the thread using pthread_create(3). In particular, the following attributes may differ:

- ? the detach state, since a joinable thread may have detached itself after creation;
- ? the stack size, which the implementation may align to a suitable boundary.
- ? and the guard size, which the implementation may round upward to a multiple of the page size, or ignore (i.e., treat as 0), if the application is allocating its own stack.

Furthermore, if the stack address attribute was not set in the thread attributes object used to create the thread, then the returned thread attributes object will report the actual stack address that the implementation selected for the thread.

When the thread attributes object returned by pthread_getattr_np() is no longer required, it

should be destroyed using `pthread_attr_destroy(3)`.

RETURN VALUE

On success, this function returns 0; on error, it returns a nonzero error number.

ERRORS

ENOMEM Insufficient memory.

In addition, if thread refers to the main thread, then `pthread_getattr_np()` can fail because of errors from various underlying calls: `fopen(3)`, if `/proc/self/maps` can't be opened; and `getrlimit(2)`, if the `RLIMIT_STACK` resource limit is not supported.

ATTRIBUTES

For an explanation of the terms used in this section, see `attributes(7)`.

??

? Interface ? Attribute ? Value ?

??

? `pthread_getattr_np()` ? Thread safety ? MT-Safe ?

??

STANDARDS

GNU; hence the suffix `"_np"` (nonportable) in the name.

HISTORY

glibc 2.2.3.

EXAMPLES

The program below demonstrates the use of `pthread_getattr_np()`. The program creates a thread that then uses `pthread_getattr_np()` to retrieve and display its guard size, stack address, and stack size attributes. Command-line arguments can be used to set these attributes to values other than the default when creating the thread. The shell sessions below demonstrate the use of the program.

In the first run, on an x86-32 system, a thread is created using default attributes:

```
$ ulimit -s # No stack limit ==> default stack size is 2 MB
unlimited
$ ./a.out
```

Attributes of created thread:

```
Guard size      = 4096 bytes
Stack address    = 0x40196000 (EOS = 0x40397000)
Stack size      = 0x201000 (2101248) bytes
```

In the following run, we see that if a guard size is specified, it is rounded up to the next multiple of the system page size (4096 bytes on x86-32):

```
$ ./a.out -g 4097
```

Thread attributes object after initializations:

Guard size = 4097 bytes

Stack address = (nil)

Stack size = 0x0 (0) bytes

Attributes of created thread:

Guard size = 8192 bytes

Stack address = 0x40196000 (EOS = 0x40397000)

Stack size = 0x201000 (2101248) bytes

In the last run, the program manually allocates a stack for the thread. In this case, the guard size attribute is ignored.

```
$ ./a.out -g 4096 -s 0x8000 -a
```

Allocated thread stack at 0x804d000

Thread attributes object after initializations:

Guard size = 4096 bytes

Stack address = 0x804d000 (EOS = 0x8055000)

Stack size = 0x8000 (32768) bytes

Attributes of created thread:

Guard size = 0 bytes

Stack address = 0x804d000 (EOS = 0x8055000)

Stack size = 0x8000 (32768) 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_stack_related_attributes(pthread_attr_t *attr, char *prefix)
```

```

{
    int s;

    size_t stack_size, guard_size;

    void *stack_addr;

    s = pthread_attr_getguardsize(attr, &guard_size);

    if (s != 0)

        errc(EXIT_FAILURE, s, "pthread_attr_getguardsize");

    printf("%sGuard size      = %zu bytes\n", prefix, guard_size);

    s = pthread_attr_getstack(attr, &stack_addr, &stack_size);

    if (s != 0)

        errc(EXIT_FAILURE, s, "pthread_attr_getstack");

    printf("%sStack address    = %p", prefix, stack_addr);

    if (stack_size > 0)

        printf(" (EOS = %p)", (char *) stack_addr + stack_size);

    printf("\n");

    printf("%sStack size      = %#zx (%zu) bytes\n",
           prefix, stack_size, stack_size);
}

static void
display_thread_attributes(pthread_t thread, char *prefix)
{
    int s;

    pthread_attr_t attr;

    s = pthread_getattr_np(thread, &attr);

    if (s != 0)

        errc(EXIT_FAILURE, s, "pthread_getattr_np");

    display_stack_related_attributes(&attr, prefix);

    s = pthread_attr_destroy(&attr);

    if (s != 0)

        errc(EXIT_FAILURE, s, "pthread_attr_destroy");
}

static void *      /* Start function for thread we create */
thread_start(void *arg)

```

```

{
    printf("Attributes of created thread:\n");
    display_thread_attributes(pthread_self(), "\t");
    exit(EXIT_SUCCESS);    /* Terminate all threads */
}

static void
usage(char *pname, char *msg)
{
    if (msg != NULL)
        fputs(msg, stderr);

    fprintf(stderr, "Usage: %s [-s stack-size [-a]]"
        " [-g guard-size]\n", pname);
    fprintf(stderr, "\t\t-a means program should allocate stack\n");
    exit(EXIT_FAILURE);
}

static pthread_attr_t * /* Get thread attributes from command line */
get_thread_attributes_from_cl(int argc, char *argv[],
                             pthread_attr_t *attrp)
{
    int s, opt, allocate_stack;
    size_t stack_size, guard_size;
    void *stack_addr;
    pthread_attr_t *ret_attrp = NULL; /* Set to attrp if we initialize
                                     a thread attributes object */

    allocate_stack = 0;
    stack_size = -1;
    guard_size = -1;
    while ((opt = getopt(argc, argv, "ag:s:")) != -1) {
        switch (opt) {
            case 'a': allocate_stack = 1; break;
            case 'g': guard_size = strtoul(optarg, NULL, 0); break;
            case 's': stack_size = strtoul(optarg, NULL, 0); break;
            default: usage(argv[0], NULL);

```

```

    }
}

if (allocate_stack && stack_size == -1)
    usage(argv[0], "Specifying -a without -s makes no sense\n");

if (argc > optind)
    usage(argv[0], "Extraneous command-line arguments\n");

if (stack_size != -1 || guard_size > 0) {
    ret_attrp = attrp;

    s = pthread_attr_init(attrp);

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

if (stack_size != -1) {
    if (!allocate_stack) {
        s = pthread_attr_setstacksize(attrp, stack_size);

        if (s != 0)
            errc(EXIT_FAILURE, s, "pthread_attr_setstacksize");
    } else {
        s = posix_memalign(&stack_addr, sysconf(_SC_PAGESIZE),
                           stack_size);

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

        printf("Allocated thread stack at %p\n\n", stack_addr);

        s = pthread_attr_setstack(attrp, stack_addr, stack_size);

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

if (guard_size != -1) {
    s = pthread_attr_setguardsize(attrp, guard_size);

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

```

```

    return ret_attrp;
}

int
main(int argc, char *argv[])
{
    int s;

    pthread_t thr;

    pthread_attr_t attr;

    pthread_attr_t *attrp = NULL; /* Set to &attr if we initialize
                                   a thread attributes object */

    attrp = get_thread_attributes_from_cl(argc, argv, &attr);
    if (attrp != NULL) {
        printf("Thread attributes object after initializations:\n");
        display_stack_related_attributes(attrp, "\t");
        printf("\n");
    }

    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_getaffinity_np\(3\)](#), [pthread_attr_getdetachstate\(3\)](#),
[pthread_attr_getguardsize\(3\)](#), [pthread_attr_getinheritsched\(3\)](#),
[pthread_attr_getschedparam\(3\)](#), [pthread_attr_getschedpolicy\(3\)](#), [pthread_attr_getscope\(3\)](#),
[pthread_attr_getstack\(3\)](#), [pthread_attr_getstackaddr\(3\)](#), [pthread_attr_getstacksize\(3\)](#),
[pthread_attr_init\(3\)](#), [pthread_create\(3\)](#), [pthreads\(7\)](#)