

NAME

`pthread_attr_setguardsize`, `pthread_attr_getguardsize` – set/get guard size attribute in thread attributes object

SYNOPSIS

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

```
int pthread_attr_setguardsize(pthread_attr_t *attr, size_t guardsize);
```

```
int pthread_attr_getguardsize(const pthread_attr_t *attr, size_t *guardsize);
```

Compile and link with `-pthread`.

DESCRIPTION

The `pthread_attr_setguardsize()` function sets the guard size attribute of the thread attributes object referred to by *attr* to the value specified in *guardsize*.

If *guardsize* is greater than 0, then for each new thread created using *attr* the system allocates an additional region of at least *guardsize* bytes at the end of the thread's stack to act as the guard area for the stack (but see BUGS).

If *guardsize* is 0, then new threads created with *attr* will not have a guard area.

The default guard size is the same as the system page size.

If the stack address attribute has been set in *attr* (using `pthread_attr_setstack(3)` or `pthread_attr_setstackaddr(3)`), meaning that the caller is allocating the thread's stack, then the guard size attribute is ignored (i.e., no guard area is created by the system): it is the application's responsibility to handle stack overflow (perhaps by using `mprotect(2)` to manually define a guard area at the end of the stack that it has allocated).

The `pthread_attr_getguardsize()` function returns the guard size attribute of the thread attributes object referred to by *attr* in the buffer pointed to by *guardsize*.

RETURN VALUE

On success, these functions return 0; on error, they return a nonzero error number.

ERRORS

POSIX.1 documents an **EINVAL** error if *attr* or *guardsize* is invalid. On Linux these functions always succeed (but portable and future-proof applications should nevertheless handle a possible error return).

VERSIONS

These functions are provided by glibc since version 2.1.

ATTRIBUTES

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

Interface	Attribute	Value
<code>pthread_attr_setguardsize()</code> , <code>pthread_attr_getguardsize()</code>	Thread safety	MT-Safe

CONFORMING TO

POSIX.1-2001, POSIX.1-2008.

NOTES

A guard area consists of virtual memory pages that are protected to prevent read and write access. If a thread overflows its stack into the guard area, then, on most hard architectures, it receives a **SIGSEGV** signal, thus notifying it of the overflow. Guard areas start on page boundaries, and the guard size is internally rounded up to the system page size when creating a thread. (Nevertheless, `pthread_attr_getguardsize()` returns the guard size that was set by `pthread_attr_setguardsize()`.)

Setting a guard size of 0 may be useful to save memory in an application that creates many threads and knows that stack overflow can never occur.

Choosing a guard size larger than the default size may be necessary for detecting stack overflows if a thread allocates large data structures on the stack.

BUGS

As at glibc 2.8, the NPTL threading implementation includes the guard area within the stack size allocation, rather than allocating extra space at the end of the stack, as POSIX.1 requires. (This can result in an **EINVAL** error from **pthread_create(3)** if the guard size value is too large, leaving no space for the actual stack.)

The obsolete LinuxThreads implementation did the right thing, allocating extra space at the end of the stack for the guard area.

EXAMPLE

See **pthread_getattr_np(3)**.

SEE ALSO

mmap(2), **mprotect(2)**, **pthread_attr_init(3)**, **pthread_attr_setstack(3)**, **pthread_attr_setstacksize(3)**, **pthread_create(3)**, **pthreads(7)**

COLOPHON

This page is part of release 5.05 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.