Rocky Enterprise Linux 9.2 Manual Pages on command 'sigaltstack.2'

C:\>man sigaltstack.2

SIGALTSTACK(2) Linux Programmer's Manual SIGALTSTACK(2)

NAME

sigaltstack - set and/or get signal stack context

SYNOPSIS

#include <signal.h>

int sigaltstack(const stack_t *ss, stack_t *old_ss);

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

sigaltstack():

_XOPEN_SOURCE >= 500

|| /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200809L

|| /* Glibc versions <= 2.19: */ _BSD_SOURCE

DESCRIPTION

sigaltstack() allows a process to define a new alternate signal stack and/or re? trieve the state of an existing alternate signal stack. An alternate signal stack is used during the execution of a signal handler if the establishment of that han? dler (see sigaction(2)) requested it.

The normal sequence of events for using an alternate signal stack is the following:

1. Allocate an area of memory to be used for the alternate signal stack.

2. Use sigaltstack() to inform the system of the existence and location of the al? ternate signal stack.

3. When establishing a signal handler using sigaction(2), inform the system that the signal handler should be executed on the alternate signal stack by specify?

ing the SA_ONSTACK flag.

The ss argument is used to specify a new alternate signal stack, while the old_ss argument is used to retrieve information about the currently established signal stack. If we are interested in performing just one of these tasks, then the other argument can be specified as NULL.

The stack_t type used to type the arguments of this function is defined as follows:

```
typedef struct {
    void  *ss_sp;     /* Base address of stack */
    int    ss_flags;  /* Flags */
    size_t ss_size;   /* Number of bytes in stack */
} stack_t;
```

To establish a new alternate signal stack, the fields of this structure are set as follows:

ss.ss_flags

This field contains either 0, or the following flag:

SS_AUTODISARM (since Linux 4.7)

Clear the alternate signal stack settings on entry to the signal han? dler. When the signal handler returns, the previous alternate signal stack settings are restored.

This flag was added in order make it safe to switch away from the signal handler with swapcontext(3). Without this flag, a subse? quently handled signal will corrupt the state of the switched-away signal handler. On kernels where this flag is not supported, sigalt? stack() fails with the error EINVAL when this flag is supplied.

ss.ss_sp

This field specifies the starting address of the stack. When a signal han? dler is invoked on the alternate stack, the kernel automatically aligns the address given in ss.ss_sp to a suitable address boundary for the underlying hardware architecture.

ss.ss_size

This field specifies the size of the stack. The constant SIGSTKSZ is de? fined to be large enough to cover the usual size requirements for an alter? nate signal stack, and the constant MINSIGSTKSZ defines the minimum size re?

quired to execute a signal handler.

To disable an existing stack, specify ss.ss_flags as SS_DISABLE. In this case, the kernel ignores any other flags in ss.ss_flags and the remaining fields in ss.

If old_ss is not NULL, then it is used to return information about the alternate signal stack which was in effect prior to the call to sigaltstack(). The old_ss.ss_sp and old_ss.ss_size fields return the starting address and size of that stack. The old_ss.ss_flags may return either of the following values:

SS_ONSTACK

The process is currently executing on the alternate signal stack. (Note that it is not possible to change the alternate signal stack if the process is currently executing on it.)

SS_DISABLE

The alternate signal stack is currently disabled.

Alternatively, this value is returned if the process is currently executing on an alternate signal stack that was established using the SS_AUTODISARM flag. In this case, it is safe to switch away from the signal handler with swapcontext(3). It is also possible to set up a different alternative sig? nal stack using a further call to sigaltstack().

SS_AUTODISARM

The alternate signal stack has been marked to be autodisarmed as described above.

By specifying ss as NULL, and old_ss as a non-NULL value, one can obtain the cur? rent settings for the alternate signal stack without changing them.

RETURN VALUE

sigaltstack() returns 0 on success, or -1 on failure with errno set to indicate the error.

ERRORS

EFAULT Either ss or old_ss is not NULL and points to an area outside of the process's address space.

EINVAL ss is not NULL and the ss_flags field contains an invalid flag.

ENOMEM The specified size of the new alternate signal stack ss.ss_size was less than MINSIGSTKSZ.

EPERM An attempt was made to change the alternate signal stack while it was active

(i.e., the process was already executing on the current alternate signal stack).

ATTRIBUTES

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

Interface	Attribute	Value
sigaltstack()	Thread safety	MT-Safe

CONFORMING TO

POSIX.1-2001, POSIX.1-2008, SUSv2, SVr4.

The SS_AUTODISARM flag is a Linux extension.

NOTES

The most common usage of an alternate signal stack is to handle the SIGSEGV signal that is generated if the space available for the normal process stack is exhausted: in this case, a signal handler for SIGSEGV cannot be invoked on the process stack; if we wish to handle it, we must use an alternate signal stack.

Establishing an alternate signal stack is useful if a process expects that it may exhaust its standard stack. This may occur, for example, because the stack grows so large that it encounters the upwardly growing heap, or it reaches a limit estab? lished by a call to setrlimit(RLIMIT_STACK, &rlim). If the standard stack is ex? hausted, the kernel sends the process a SIGSEGV signal. In these circumstances the only way to catch this signal is on an alternate signal stack.

On most hardware architectures supported by Linux, stacks grow downward. sigalt? stack() automatically takes account of the direction of stack growth.

Functions called from a signal handler executing on an alternate signal stack will also use the alternate signal stack. (This also applies to any handlers invoked for other signals while the process is executing on the alternate signal stack.)

Unlike the standard stack, the system does not automatically extend the alternate signal stack. Exceeding the allocated size of the alternate signal stack will lead to unpredictable results.

A successful call to execve(2) removes any existing alternate signal stack. A child process created via fork(2) inherits a copy of its parent's alternate signal

stack settings.

sigaltstack() supersedes the older sigstack() call. For backward compatibility, glibc also provides sigstack(). All new applications should be written using sigaltstack().

History

4.2BSD had a sigstack() system call. It used a slightly different struct, and had the major disadvantage that the caller had to know the direction of stack growth.

EXAMPLE

The following code segment demonstrates the use of sigaltstack() (and sigaction(2)) to install an alternate signal stack that is employed by a handler for the SIGSEGV signal:

```
stack_t ss;
ss.ss_sp = malloc(SIGSTKSZ);
if (ss.ss_sp == NULL) {
    perror("malloc");
    exit(EXIT_FAILURE);
}
ss.ss_size = SIGSTKSZ;
ss.ss_flags = 0;
if (sigaltstack(&ss, NULL) == -1) {
    perror("sigaltstack");
    exit(EXIT_FAILURE);
}
sa.sa_flags = SA_ONSTACK;
sa.sa_handler = handler();      /* Address of a signal handler */
sigemptyset(&sa.sa_mask);
if (sigaction(SIGSEGV, &sa, NULL) == -1) {
    perror("sigaction");
    exit(EXIT_FAILURE);
}
```

BUGS

In Linux 2.2 and earlier, the only flag that could be specified in ss.sa_flags was SS_DISABLE. In the lead up to the release of the Linux 2.4 kernel, a change was

made to allow sigaltstack() to allow ss.ss_flags==SS_ONSTACK with the same meaning as ss.ss_flags==0 (i.e., the inclusion of SS_ONSTACK in ss.ss_flags is a no-op).

On other implementations, and according to POSIX.1, SS_ONSTACK appears only as a reported flag in old_ss.ss_flags. On Linux, there is no need ever to specify SS_ONSTACK in ss.ss_flags, and indeed doing so should be avoided on portability grounds: various other systems give an error if SS_ONSTACK is specified in ss.ss_flags.

SEE ALSO

execve(2), setrlimit(2), sigaction(2), siglongjmp(3), sigsetjmp(3), signal(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/.