NAME

sigaction, signal − manage signal state and handlers

SYNOPSIS

#include <signal.h>

int sigaction(int *sig***, const struct sigaction ***act***, struct sigaction ***oact***)**
void (*signal(int *sig***, void (***handler***)(int)))(int);**

DESCRIPTION

Sigaction() is used to examine, set, or modify the attributes of a signal. The argument *sig* is the signal in question. The *act* argument points to a structure containing the new attributes of the signal, the structure pointed to by *oact* will receive the old attributes that were in effect before the call.

The *act* and *oact* arguments may be **NULL** to indicate that either no new attributes are to be set, or that the old attributes are not of interest.

The structure containing the signal attributes is defined in <signal.h> and looks like this:

```
struct sigaction {
      void        (*sa_handler)(int sig);
      sigset_t    sa_mask;
      int         sa_flags;
};
```

The **sa_handler** field contains the address of a signal handler, a function that is called when the process is signalled, or one of these special constants:

SIG_DFL Default signal handling is to be performed. This usually means that the process is killed, but some signals may be ignored by default.

SIG_IGN Ignore the signal.

The **sa_mask** field indicates a set of signals that must be blocked when the signal is being handled. Whether the signal *sig* itself is blocked when being handled is not controlled by this mask. The mask is of a "signal set" type that is to be manipulated by the **sigset**(3) functions.

How the signal is handled precisely is specified by bits in **sa_flags**. If none of the flags is set then the handler is called when the signal arrives. The signal is blocked during the call to the handler, and unblocked when the handler returns. A system call that is interrupted returns **−1** with **errno** set to **EINTR**. The following bit flags can be set to modify this behaviour:

SA_RESETHAND
Reset the signal handler to **SIG_DFL** when the signal is caught.

SA_NODEFER Do not block the signal on entry to the handler.

SA_COMPAT Handle the signal in a way that is compatible with the the old **signal()** call.

The old **signal()** signal system call sets a signal handler for a given signal and returns the old signal handler. No signals are blocked, the flags are **SA_RESETHAND | SA_NODEFER | SA_COMPAT**. New code should not use **signal()**. Note that **signal()** and all of the **SA_*** flags are Minix extensions.

Signal handlers are reset to **SIG_DFL** on an **execve**(2). Signals that are ignored stay ignored.

Signals

Minix knows about the following signals:

signal	num	notes	description
SIGHUP	1	k	Hangup
SIGINT	2	k	Interrupt (usually DEL or CTRL−C)
SIGQUIT	3	kc	Quit (usually CTRL−\)
SIGILL	4	kc	Illegal instruction
SIGTRAP	5	xkc	Trace trap
SIGABRT	6	kc	Abort program

SIGFPE	8	k	Floating point exception
SIGKILL	9	k	Kill
SIGUSR1	10	k	User defined signal #1
SIGSEGV	11	kc	Segmentation fault
SIGUSR2	12	k	User defined signal #2
SIGPIPE	13	k	Write to a pipe with no reader
SIGALRM	14	k	Alarm clock
SIGTERM	15	k	Terminate (default for kill(1))
SIGCHLD	17	pvi	Child process terminated
SIGCONT	18	p	Continue if stopped
SIGSTOP	19	ps	Stop signal
SIGTSTP	20	ps	Interactive stop signal
SIGTTIN	21	ps	Background read
SIGTTOU	22	ps	Background write
SIGWINCH	23	xvi	Window size change

The letters in the notes column indicate:

k The process is killed if the signal is not caught.

c The signal causes a core dump.

i The signal is ignored if not caught.

v Only Minix-vmd implements this signal.

x Minix extension, not defined by POSIX.

p These signals are not implemented, but POSIX requires that they are defined.

s The process should be stopped, but is killed instead.

The **SIGKILL** signal cannot be caught or ignored. The **SIGILL** and **SIGTRAP** signals cannot be automatically reset. The system silently enforces these restrictions. This may or may not be reflected by the attributes of these signals and the signal masks.

Types

POSIX prescribes that <sys/types.h> has the following definition:

typedef int (*sighandler_t)(int)

With this type the following declarations can be made:

sighandler_t sa_handler;
sighandler_t signal(int *sig***, sighandler_t** *handler***);**

This may help you to understand the earlier declarations better. The **sighandler_t** type is also very useful in old style C code that is compiled by a compiler for standard C.

SEE ALSO

kill(1), **kill**(2), **pause**(2), **sigprocmask**(2), **sigsuspend**(2), **sigpending**(2), **sigset**(3).

DIAGNOSTICS

Sigaction() returns **0** on success or **−1** on error. **Signal()** returns the old handler on success or **SIG_ERR** on error. The error code may be:

EINVAL Bad signal number.

EFAULT Bad *act* or *oact* addresses.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)