NAME

sigvec – software signal facilities

SYNOPSIS

/usr/ucb/cc [*flag* ...] *file* ...

#include <signal.h>

int sigvec(*sig, nvec, ovec***)**
int *sig***;**
struct sigvec **nvec*, **ovec***;**

DESCRIPTION

The system defines a set of signals that may be delivered to a process. Signal delivery resembles the occurrence of a hardware interrupt: the signal is blocked from further occurrence, the current process context is saved, and a new one is built. A process may specify a *handler* to which a signal is delivered, or specify that a signal is to be *blocked* or *ignored*. A process may also specify that a default action is to be taken by the system when a signal occurs. Normally, signal handlers execute on the current stack of the process. This may be changed, on a per-handler basis, so that signals are taken on a special *signal stack*.

All signals have the same *priority*. Signal routines execute with the signal that caused their invocation to be *blocked*, but other signals may yet occur. A global *signal mask* defines the set of signals currently blocked from delivery to a process. The signal mask for a process is initialized from that of its parent (normally 0). It may be changed with a **sigblock( )** or **sigsetmask( )** call, or when a signal is delivered to the process.

A process may also specify a set of *flags* for a signal that affect the delivery of that signal.

When a signal condition arises for a process, the signal is added to a set of signals pending for the process. If the signal is not currently *blocked* by the process then it is delivered to the process. When a signal is delivered, the current state of the process is saved, a new signal mask is calculated (as described below), and the signal handler is invoked. The call to the handler is arranged so that if the signal handling routine returns normally the process will resume execution in the context from before the signal's delivery. If the process wishes to resume in a different context, then it must arrange to restore the previous context itself.

When a signal is delivered to a process a new signal mask is installed for the duration of the process' signal handler (or until a **sigblock( )** or **sigsetmask( )** call is made). This mask is formed by taking the current signal mask, adding the signal to be delivered, and ORing in the signal mask associated with the handler to be invoked.

The action to be taken when the signal is delivered is specified by a **sigvec( )** structure, which includes the following members:

```
void      (*sv_handler)( );    /* signal handler */
int       sv_mask;             /* signal mask to apply */
int       sv_flags;            /* see signal options */

#define   SV_ONSTACK           /* take signal on signal stack */
#define   SV_INTERRUPT         /* do not restart system on signal return */
#define   SV_RESETHAND         /* reset handler to SIG_DFL when signal taken */
```

If the **SV_ONSTACK** bit is set in the flags for that signal, the system will deliver the signal to the process on the signal stack specified with **sigstack**(3B) rather than delivering the signal on the current stack.

If *nvec* is not a **NULL** pointer, **sigvec( )** assigns the handler specified by **sv_handler( )**, the mask specified by **sv_mask( )**, and the flags specified by **sv_flags( )** to the specified signal. If *nvec* is a **NULL** pointer, **sigvec( )** does not change the handler, mask, or flags for the specified signal.

The mask specified in *nvec* is not allowed to block **SIGKILL**, **SIGSTOP**, or **SIGCONT**. The system enforces this restriction silently.

If *ovec* is not a **NULL** pointer, the handler, mask, and flags in effect for the signal before the call to **sigvec( )** are returned to the user. A call to **sigvec( )** with *nvec* a **NULL** pointer and *ovec* not a **NULL** pointer can be used to determine the handling information currently in effect for a signal without changing that information.

The following is a list of all signals with names as in the include file **<signal.h>**:

SIGHUP		hangup
SIGINT		interrupt
SIGQUIT	*	quit
SIGILL	*	illegal instruction
SIGTRAP	*	trace trap
SIGABRT	*	abort (generated by **abort**(3C) routine)
SIGEMT	*	emulator trap
SIGFPE	*	arithmetic exception
SIGKILL		kill (cannot be caught, blocked, or ignored)
SIGBUS	*	bus error
SIGSEGV	*	segmentation violation
SIGSYS	*	bad argument to function
SIGPIPE		write on a pipe or other socket with no one to read it
SIGALRM		alarm clock
SIGTERM		software termination signal
SIGURG	•	urgent condition present on socket
SIGSTOP	†	stop (cannot be caught, blocked, or ignored)
SIGTSTP	†	stop signal generated from keyboard
SIGCONT	•	continue after stop (cannot be blocked)
SIGCHLD	•	child status has changed
SIGTTIN	†	background read attempted from control terminal
SIGTTOU	†	background write attempted to control terminal
SIGIO	•	I/O is possible on a descriptor (see **fcntl**(2))
SIGXCPU		cpu time limit exceeded (see **getrlimit**(2))
SIGXFSZ		file size limit exceeded (see **getrlimit**(2))
SIGVTALRM		virtual time alarm (see **setitimer( )** on **getitimer**(2))
SIGPROF		profiling timer alarm (see **setitimer( )** on **getitimer**(2))

SIGWINCH	●	window changed (see **termio**(7I))
SIGLOST	*	resource lost (see **lockd**(1M))
SIGUSR1		user-defined signal 1
SIGUSR2		user-defined signal 2

The starred signals in the list above cause a core image if not caught or ignored.

Once a signal handler is installed, it remains installed until another **sigvec( )** call is made, or an **execve**(2) is performed, unless the **SV_RESETHAND** bit is set in the flags for that signal. In that case, the value of the handler for the caught signal will be set to **SIG_DFL** before entering the signal-catching function, unless the signal is **SIGILL**, **SIGPWR**, or **SIGTRAP**. Also, if this bit is set, the bit for that signal in the signal mask will not be set; unless the signal mask associated with that signal blocks that signal, further occurrences of that signal will not be blocked. The **SV_RESETHAND** flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

The default action for a signal may be reinstated by setting the signal's handler to **SIG_DFL**; this default is termination except for signals marked with ● or †. Signals marked with ● are discarded if the action is **SIG_DFL**; signals marked with † cause the process to stop. If the process is terminated, a "core image" will be made in the current working directory of the receiving process if the signal is one for which an asterisk appears in the above list (see **core**(4)).

If the handler for that signal is **SIG_IGN**, the signal is subsequently ignored, and pending instances of the signal are discarded.

If a caught signal occurs during certain functions, the call is normally restarted. The call can be forced to terminate prematurely with an **EINTR** error return by setting the **SV_INTERRUPT** bit in the flags for that signal. The **SV_INTERRUPT** flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed. The affected functions are **read**(2) or **write**(2) on a slow device (such as a terminal or pipe or other socket, but not a file) and during a **wait**(2).

After a **fork**(2) or **vfork**(2) the child inherits all signals, the signal mask, the signal stack, and the restart/interrupt and reset-signal-handler flags.

The **execve**(2) call resets all caught signals to default action and resets all signals to be caught on the user stack. Ignored signals remain ignored; the signal mask remains the same; signals that interrupt functions continue to do so.

The accuracy of *addr* is machine dependent. For example, certain machines may supply an address that is on the same page as the address that caused the fault. If an appropriate *addr* cannot be computed it will be set to **SIG_NOADDR**.

RETURN VALUES

A 0 value indicates that the call succeeded. A −1 return value indicates that an error occurred and **errno** is set to indicate the reason.

ERRORS

sigvec( ) will fail and no new signal handler will be installed if one of the following occurs:

EFAULT Either *nvec* or *ovec* is not a **NULL** pointer and points to memory that is not a valid part of the process address space.

EINVAL *sig* is not a valid signal number, or, **SIGKILL**, or **SIGSTOP**.

SEE ALSO

intro(2), **exec**(2), **fcntl**(2), **fork**(2), **getitimer**(2), **getrlimit**(2), **ioctl**(2), **kill**(2), **ptrace**(2), **read**(2), **umask**(2), **vfork**(2), **wait**(2), **write**(2), **setjmp**(3C) **sigblock**(3B), **sigstack**(3B), **signal**(3B), **wait**(3B), **signal**(3C), **core**(4), **streamio**(7I), **termio**(7I)

NOTES

Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

SIGPOLL is a synonym for **SIGIO**. A **SIGIO** will be issued when a file descriptor corresponding to a STREAMS (see **intro**(2)) file has a "selectable" event pending. Unless that descriptor has been put into asynchronous mode (see **fcntl**(2)), a process may specifically request that this signal be sent using the **I_SETSIG ioctl**(2) call (see **streamio**(7I)). Otherwise, the process will never receive **SIGPOLLs0**.

The handler routine can be declared:

void handler(int *sig*, **int** *code*, **struct sigcontext** **scp*, **char** **addr*);

Here *sig* is the signal number; *code* is a parameter of certain signals that provides additional detail; *scp* is a pointer to the **sigcontext** structure (defined in **signal.h**), used to restore the context from before the signal; and *addr* is additional address information.

The signals **SIGKILL**, **SIGSTOP**, and **SIGCONT** cannot be ignored.