NAME

sigvec - software signal facilities

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

```
#include <signal.h>

struct sigvec {
        void    (*sv_handler)();
                int     sv_mask;
                int     sv_flags;
};
int
sigvec(int sig, struct sigvec *vec, struct sigvec *ovec);
```

DESCRIPTION

This interface is made obsolete by sigaction(2).

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 *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(2) or sigsetmask(2) call, or when a signal is delivered to the process.

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(2) or sigsetmask(2) call is made). This mask is formed by taking the current signal mask, adding the signal to be delivered, and *or'*ing in the signal mask associated with the handler to be invoked.

The **sigvec**() function assigns a handler for a specific signal. If *vec* is non-zero, it specifies a handler routine and mask to be used when delivering the specified signal. Further, if the SV_ONSTACK bit is set in *sv_flags*, the system will deliver the signal to the process on a *signal stack*, specified with sigaltstack(2). If *ovec* is non-zero, the previous handling information for the signal is returned to the user.

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

NAME	Default Action	Description
SIGHUP	terminate process	terminal line hangup
SIGINT	terminate process	interrupt program
SIGQUIT	create core image	quit program
SIGILL	create core image	illegal instruction
SIGTRAP	create core image	trace trap
SIGABRT	create core image	abort(3) call (formerly SIGIOT)
SIGEMT	create core image	emulate instruction executed
SIGFPE	create core image	floating-point exception
SIGKILL	terminate process	kill program
SIGBUS	create core image	bus error
SIGSEGV	create core image	segmentation violation
SIGSYS	create core image	non-existent system call invoked
SIGPIPE	terminate process	write on a pipe with no reader
SIGALRM	terminate process	real-time timer expired
SIGTERM	terminate process	software termination signal
SIGURG	discard signal	urgent condition present on socket
SIGSTOP	stop process	stop (cannot be caught or ignored)
SIGTSTP	stop process	stop signal generated from keyboard
SIGCONT	discard signal	continue after stop
SIGCHLD	discard signal	child status has changed
SIGTTIN	stop process	background read attempted from control terminal
SIGTTOU	stop process	background write attempted to control terminal
SIGIO	discard signal	I/O is possible on a descriptor (see fcntl(2))
SIGXCPU	terminate process	cpu time limit exceeded (see setrlimit(2))

SIGXFSZ	terminate process	file size limit exceeded (see setrlimit(2))
SIGVTALRM	terminate process	virtual time alarm (see setitimer(2))
SIGPROF	terminate process	profiling timer alarm (see setitimer(2))
SIGWINCH	discard signal	Window size change
SIGINFO	discard signal	status request from keyboard
SIGUSR1	terminate process	User defined signal 1
SIGUSR2	terminate process	User defined signal 2

Once a signal handler is installed, it remains installed until another **sigvec**() call is made, or an execve(2) is performed. A signal-specific default action may be reset by setting *sv_handler* to SIG_DFL. The defaults are process termination, possibly with core dump; no action; stopping the process; or continuing the process. See the above signal list for each signal's default action. If *sv_handler* is SIG_IGN current and pending instances of the signal are ignored and discarded.

If a signal is caught during the system calls listed below, 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 *sv_flags*. The affected system calls include read(2), write(2), sendto(2), recvfrom(2), sendmsg(2) and recvmsg(2) on a communications channel or a slow device (such as a terminal, but not a regular file) and during a wait(2) or ioctl(2). However, calls that have already committed are not restarted, but instead return a partial success (for example, a short read count).

After a fork(2) or vfork(2) all signals, the signal mask, the signal stack, and the restart/interrupt flags are inherited by the child.

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

NOTES

The mask specified in *vec* is not allowed to block SIGKILL or SIGSTOP. This is done silently by the system.

The SV_INTERRUPT flag is not available in 4.2BSD, hence it should not be used if backward compatibility is needed.

RETURN VALUES

The **sigvec**() function returns the value 0 if successful; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

EXAMPLES

On the VAX-11 The handler routine can be declared:

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

Here *sig* is the signal number, into which the hardware faults and traps are mapped as defined below. The *code* argument is either a constant as given below or, for compatibility mode faults, the code provided by the hardware (Compatibility mode faults are distinguished from the other SIGILL traps by having PSL_CM set in the psl). The *scp* argument is a pointer to the *sigcontext* structure (defined in <*signal.h*>), used to restore the context from before the signal.

ERRORS

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

[EFAULT] Either *vec* or *ovec* points to memory that is not a valid part of the process address space.

[EINVAL] The *sig* argument is not a valid signal number.

[EINVAL] An attempt is made to ignore or supply a handler for SIGKILL or SIGSTOP.

SEE ALSO

kill(1), kill(2), ptrace(2), sigaction(2), sigaltstack(2), sigblock(2), sigpause(2), sigprocmask(2), sigsetmask(2), sigsuspend(2), setjmp(3), siginterrupt(3), signal(3), sigsetops(3), tty(4)

HISTORY

A **sigvec**() system call first appeared in 4.2BSD. It was reimplemented as a wrapper around sigaction(2) in 4.3BSD-Reno.

BUGS

This manual page is still confusing.