NAME

signal – simplified software signal facilities

SYNOPSIS

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

#include <signal.h>

void (∗signal(*sig,func***))()**
int *sig***;**
void (∗*func***)()***;*

DESCRIPTION

signal( ) is a simplified interface to the more general **sigvec**(3B) facility. Programs that use **signal( )** in preference to **sigvec( )** are more likely to be portable to all systems.

A signal is generated by some abnormal event, initiated by a user at a terminal (quit, interrupt, stop), by a program error (bus error, etc.), by request of another program (kill), or when a process is stopped because it wishes to access its control terminal while in the background (see **termio**(7I)). Signals are optionally generated when a process resumes after being stopped, when the status of child processes changes, or when input is ready at the control terminal. Most signals cause termination of the receiving process if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the **SIGKILL** and **SIGSTOP** signals, the **signal( )** call allows signals either to be ignored or to interrupt to a specified location. See **sigvec**(3B) for a complete list of the signals.

If *func* is **SIG_DFL**, the default action for signal *sig* is reinstated; this default is termination (with a core image for starred signals) 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 *func* is **SIG_IGN** the signal is subsequently ignored and pending instances of the signal are discarded. Otherwise, when the signal occurs further occurrences of the signal are automatically blocked and *func* is called.

A return from the function unblocks the handled signal and continues the process at the point it was interrupted.

If a caught signal occurs during certain functions, terminating the call prematurely, the call is automatically restarted. In particular this can occur during a **read**(2) or **write**(2) on a slow device (such as a terminal; but not a file) and during a **wait**(2).

The value of **signal( )** is the previous (or initial) value of *func* for the particular signal.

After a **fork**(2) or **vfork**(2) the child inherits all signals. An **exec**(2) resets all caught signals to the default action; ignored signals remain ignored.

RETURN VALUES

The previous action is returned on a successful call. Otherwise, −1 is returned and **errno** is set to indicate the error.

ERRORS

signal( ) will fail and no action will take place if the following occurs:

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

SEE ALSO

kill(1), **exec**(2), **fcntl**(2), **fork**(2), **getitimer**(2), **getrlimit**(2), **kill**(2), **ptrace**(2), **read**(2), **sigaction**(2), **wait**(2), **write**(2), **abort**(3C), **setjmp**(3B), **sigblock**(3B), **sigstack**(3B), **sigvec**(3B), **wait**(3B), **setjmp**(3C), **signal**(3C), **signal**(5), **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.

The handler routine, *func*, can be declared:

void handler(*signum***)**
int *signum***;**

Here *signum* is the signal number. See **sigvec**(3B) for more details.