NAME

signal − catch or ignore signals

SYNOPSIS

```
#include <signal.h>

int (*signal (sig, func))( )
int sig;
(*func)( );
```

DESCRIPTION

A signal is generated by some abnormal event, initiated either by a user at a typewriter (quit, interrupt), by a program error (bus error, etc.), or by request of another program (kill). Normally, all signals (except death of a child and power fail) cause termination of the receiving process, but a *signal* call allows them either to be ignored or to cause an interrupt to a specified location. Here is the list of signals:

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction (not reset when caught)
SIGTRAP	5*	trace trap (not reset when caught)
SIGIOT	6*	IOT instruction
SIGEMT	7*	EMT instruction
SIGFPE	8*	floating point exception
SIGKILL	9	kill (cannot be caught or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	catchable software termination signal
	16	unassigned
	17	unassigned
SIGCLD	18	death of a child
SIGPWR	19	power fail

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

If *func* is SIG_DFL, the default action for signal *sig* is reinstated; this default is termination, sometimes with a core image. If *func* is SIG_IGN, the signal is ignored. Otherwise when the signal occurs *func* will be called with the signal number as argument. A return from the function will continue the process at the point it was interrupted. Except as indicated, a signal is reset to SIG_DFL after being caught. Thus if it is desired to catch every such signal, the catching routine must issue another *signal* call.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular this can occur during a *read* or *write*(2) on a slow device (like a typewriter; but not a file); and during *pause* or *wait*(2). When such a signal occurs, the saved user status is arranged in such a way that when return from the signal-catching takes place, it will appear that the system call returned an error status. The user's program may then, if it wishes, re-execute the call.

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

After a *fork*(2) the child inherits all signals. *Exec*(2) resets all caught signals to default action.

Users should not use the signal numbers directly; instead, they should include the file **/usr/include/signal.h** as indicated above.

The default action for the death of a child signal is to ignore the signal. If *label* is odd, the signal is ignored and terminated child processes are automatically removed from the system — eliminating the necessity of doing a *wait*(2) for the terminated children.

For the power fail signal, the default action is to ignore it.

SEE ALSO

kill(1), kill(2), ptrace(2), setjmp(3C)

DIAGNOSTICS

The value **−1** is returned if the given signal is out of range.

BUGS

If a repeated signal arrives before the last one can be reset, there is no chance to catch it.

ASSEMBLER

(signal = 48.)
sys signal; sig; label
(old value in r0)

If *label* is 0, default action is reinstated. If *label* is odd, the signal is ignored. Any other even *label* specifies an address in the process where an interrupt is simulated. An RTI or RTT instruction will return from the interrupt.