NAME

signal − catch or ignore signals

SYNOPSIS

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

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

DESCRIPTION

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

1	hangup
2	interrupt
3*	quit
4*	illegal instruction (not reset when caught)
5*	trace trap (not reset when caught)
6*	IOT instruction
7*	EMT instruction
8*	floating point exception
9	kill (cannot be caught or ignored)
10*	bus error
11*	segmentation violation
12*	nonexistent system call
13	write on a pipe with no one to read it
14	alarm clock
15	software termination (catchable kill)

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

In the assembler call, if *label* is 0, the process is terminated when the signal occurs; this is the default action. 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. Except as indicated, a signal is reset to 0 after being caught. Thus if it is desired to catch every such signal, the catching routine must issue another *signal* call.

In C, if *func* is 0, the default action for signal *sig* (termination) is reinstated. If *func* is 1, the signal is ignored. If *func* is non-zero and even, it is assumed to be the address of a function entry point. When the signal occurs, the function will be called. A return from the function will continue the process at the point it was interrupted. As in the assembler call, *signal* must in general be called again to catch subsequent signals.

When a caught signal occurs during certain system calls, the call terminates prematurely. In particular, this can occur during a *read* or *write* on a slow device (like a terminal, but not a disk file), and during *wait.* 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 a characteristic error status. The user’s program may then, if it wishes, re-execute the call.

The value of the call is the old action defined for the signal.

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

SEE ALSO

kill(I), kill(II), ptrace(II), reset(III)

DIAGNOSTICS

The error bit (c-bit) is set if the given signal is out of range. In C, a −1 indicates an error.