NAME

sigaction - detailed signal management

SYNOPSIS

```
#include <signal.h>

int sigaction(int sig, const struct sigaction  *act,  struct
sigaction *oact);
```

DESCRIPTION

The sigaction() function allows the calling process to examine or specify the action to be taken on delivery of a specific signal. See signal(3HEAD) for an explanation of general signal concepts.

The sig argument specifies the signal and can be assigned any of the signals specified in signal(3HEAD) except SIGKILL and SIGSTOP. In a multithreaded process, sig cannot be SIGWAITING, SIGCANCEL, or SIGLWP.

If the argument act is not NULL, it points to a structure specifying the new action to be taken when delivering sig. If the argument oact is not NULL, it points to a structure where the action previously associated with sig is to be stored on return from sigaction().

The sigaction structure includes the following members:

```
void      (*sa_handler)();
void      (*sa_sigaction)(int, siginfo_t *, void *);
sigset_t  sa_mask;
int       sa_flags;
```

The storage occupied by sa_handler and sa_sigaction may overlap, and a standard-conforming application (see standards(5)) must not use both simultaneously.

The sa_handler member identifies the action to be associated with the specified signal, if the SA_SIGINFO flag (see below) is cleared in the sa_flags field of the sigaction structure. It may take any of the values specified in signal(3HEAD) or that of a user specified signal handler. If the SA_SIGINFO flag is set in the sa_flags field, the sa_sigaction field specifies a signal-catching function.

The sa_mask member specifies a set of signals to be blocked while the signal handler is active. On entry to the signal handler, that set of signals is added to the set of signals already being blocked when the signal is delivered. In addition, the signal that caused the handler to be executed will also be blocked, unless the SA_NODEFER flag has been specified. SIGSTOP and SIGKILL cannot be blocked (the system silently enforces this restriction).

The sa_flags member specifies a set of flags used to modify the delivery of the signal. It is formed by a logical OR of any of the following values:

SA_ONSTACK
    If set and the signal is caught, and if the thread that is chosen to processes a delivered signal has an alternate signal stack declared with sigaltstack(2), then it will process the signal on that stack. Otherwise, the signal is delivered on the thread’s normal stack.

SA_RESETHAND
    If set and the signal is caught, the disposition of the signal is reset to SIG_DFL and the signal will not be blocked on entry to the signal handler (SIGILL, SIGTRAP, and SIGPWR cannot be automatically reset when delivered; the system silently enforces this restriction).

```
     SA_NODEFER
           If set and the signal is caught, the signal  will  not
           be  automatically  blocked  by  the kernel while it is
           being caught.

     SA_RESTART
           If set and the signal is caught,  functions  that  are
           interrupted  by the execution of this signal's handler
           are transparently  restarted  by  the  system,  namely
           fcntl(2),  ioctl(2),  wait(2), waitid(2), and the fol-
           lowing  functions  on  slow  devices  like  terminals:
           getmsg()  and getpmsg() (see getmsg(2));  putmsg() and
           putpmsg()  (see  putmsg(2));  pread(),   read(),   and
           readv() (see read(2)); pwrite(), write(), and writev()
           (see write(2)); recv(), recvfrom(), and recvmsg() (see
           recv(3SOCKET));  and  send(),  sendto(), and sendmsg()
           (see send(3SOCKET). Otherwise, the function returns an
           EINTR error.

     SA_SIGINFO
           If cleared and the signal is caught, sig is passed  as
           the  only argument to the signal-catching function. If
           set and the signal is caught,   two  additional  argu-
           ments  are passed to the signal-catching function.  If
           the second argument is not equal to NULL, it points to
           a  siginfo_t  structure  containing the reason why the
           signal was generated (see siginfo(3HEAD));  the  third
           argument  points  to a ucontext_t structure containing
           the receiving process's context when  the  signal  was
           delivered (see ucontext(3HEAD)).

     SA_NOCLDWAIT
           If set and sig equals  SIGCHLD, the  system  will  not
           create  zombie  processes when children of the calling
           process exit.  If  the  calling  process  subsequently
           issues  a  wait(2), it blocks until all of the calling
           process's child processes terminate, and then  returns
           -1 with errno set to ECHILD.

     SA_NOCLDSTOP
           If set and sig equals SIGCHLD,  SIGCHLD  will  not  be
           sent  to  the calling process when its child processes
           stop or continue.

RETURN VALUES
     Upon successful completion, 0 is returned. Otherwise, -1  is
     returned,  errno  is  set  to indicate the error, and no new
     signal handler is installed.

ERRORS
     The sigaction() function will fail if:

     EINVAL
           The value of the sig argument is not  a  valid  signal
           number  or  is  equal to  SIGKILL or SIGSTOP. In addi-
           tion, if in a multithreaded process, it  is  equal  to
           SIGWAITING, SIGCANCEL, or SIGLWP.

ATTRIBUTES
     See attributes(5) for descriptions of the  following  attri-
     butes:
```

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Standard
MT-Level	Async-Signal-Safe

```
SEE ALSO
     kill(1), intro(2), exit(2), fcntl(2),  getmsg(2),  ioctl(2),
     kill(2),  pause(2), putmsg(2), read(2), sigaltstack(2), sig-
     procmask(2), sigsend(2), sigsuspend(2), wait(2),  waitid(2),
     write(2),   recv(3SOCKET),   send(3SOCKET),  siginfo(3HEAD),
     signal(3C), signal(3HEAD), sigsetops(3C),  thr_create(3THR),
     ucontext(3HEAD), attributes(5), standards(5)
```

```
NOTES
     The handler routine can be declared:

     void handler (int sig, siginfo_t *sip, ucontext_t *ucp);

     The sig argument is the signal number. The sip argument is a
     pointer  (to  space  on the stack) to a siginfo_t structure,
     which provides additional detail about the delivery  of  the
     signal. The ucp argument is a pointer (again to space on the
     stack)   to   a    ucontext_t    structure    (defined    in
     <sys/ucontext.h>) which contains the context from before the
     signal.  It is not recommended  that  ucp  be  used  by  the
     handler  to  restore  the  context  from  before  the signal
     delivery.

SunOS 5.9            Last change: 9 Jul 2002
```