Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'signal.h.0p' command

$ man signal.h.0p

signal.h(0P) POSIX Programmer's Manual signal.h(0P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

signal.h ? signals

SYNOPSIS

#include <signal.h>

DESCRIPTION

Some of the functionality described on this reference page extends the ISO C standard. Applications shall define the appropriate feature test macro (see the System Interfaces volume of POSIX.1?2017, Section 2.2, The Compilation Environment) to enable the visibility of these symbols in this header.

The <signal.h> header shall define the following macros, which shall expand to constant expressions with distinct values that have a type compatible with the second argument to, and the return value of, the signal() function, and whose values shall compare unequal to the ad? dress of any declarable function.

SIG_DFL Request for default signal handling.

SIG_ERR Return value from signal() in case of error.

SIG_HOLD Request that signal be held.

SIG_IGN Request that signal be ignored.

The <signal.h> header shall define the pthread_t, size_t, and uid_t types as described in <sys/types.h>.

The <signal.h> header shall define the timespec structure as described in <time.h>.

The <signal.h> header shall define the following data types:

sig_atomic_t Possibly volatile-qualified integer type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts.

sigset_t Integer or structure type of an object used to represent sets of signals.

pid_t As described in <sys/types.h>.

The <signal.h> header shall define the pthread_attr_t type as described in <sys/types.h>.

The <signal.h> header shall define the sigevent structure, which shall include at least the following members:

```
int             sigev_notify            Notification type.
int             sigev_signo             Signal number.
union sigval    sigev_value             Signal value.
void            (*sigev_notify_function)(union sigval)
                                        Notification function.
pthread_attr_t *sigev_notify_attributes  Notification attributes.
```

The <signal.h> header shall define the following symbolic constants for the values of sigev_notify:

SIGEV_NONE No asynchronous notification is delivered when the event of interest occurs.

SIGEV_SIGNAL A queued signal, with an application-defined value, is generated when the event of interest occurs.

SIGEV_THREAD A notification function is called to perform notifica? tion.

The sigval union shall be defined as:

```
int    sival_int    Integer signal value.
```

void *sival_ptr Pointer signal value.

The <signal.h> header shall declare the SIGRTMIN and SIGRTMAX macros, which shall expand to positive integer expressions with type int, but which need not be constant expressions. These macros specify a range of signal numbers that are reserved for application use and for which the realtime signal behavior specified in this volume of POSIX.1?2017 is supported. The signal numbers in this range do not overlap any of the signals specified in the following table.

The range SIGRTMIN through SIGRTMAX inclusive shall include at least {RTSIG_MAX} signal numbers.

It is implementation-defined whether realtime signal behavior is sup? ported for other signals.

The <signal.h> header shall define the following macros that are used to refer to the signals that occur in the system. Signals defined here begin with the letters SIG followed by an uppercase letter. The macros shall expand to positive integer constant expressions with type int and distinct values. The value 0 is reserved for use as the null signal (see kill()). Additional implementation-defined signals may occur in the system.

The ISO C standard only requires the signal names SIGABRT, SIGFPE, SIG? ILL, SIGINT, SIGSEGV, and SIGTERM to be defined. An implementation need not generate any of these six signals, except as a result of explicit use of interfaces that generate signals, such as raise(), kill(), the General Terminal Interface (see Section 11.1.9, Special Characters), and the kill utility, unless otherwise stated (see, for example, the System Interfaces volume of POSIX.1?2017, Section 2.8.3.3, Memory Pro? tection).

The following signals shall be supported on all implementations (de? fault actions are explained below the table):

Signal	Default Action	Description
SIGABRT	A	Process abort signal.

?SIGALRM ? T ? Alarm clock. ?

?SIGBUS ? A ? Access to an undefined portion of a memory object. ?

?SIGCHLD ? I ? Child process terminated, stopped, ?

? ? ? or continued. ?

?SIGCONT ? C ? Continue executing, if stopped. ?

?SIGFPE ? A ? Erroneous arithmetic operation. ?

?SIGHUP ? T ? Hangup. ?

?SIGILL ? A ? Illegal instruction. ?

?SIGINT ? T ? Terminal interrupt signal. ?

?SIGKILL ? T ? Kill (cannot be caught or ignored). ?

?SIGPIPE ? T ? Write on a pipe with no one to read it. ?

?SIGQUIT ? A ? Terminal quit signal. ?

?SIGSEGV ? A ? Invalid memory reference. ?

?SIGSTOP ? S ? Stop executing (cannot be caught or ignored). ?

?SIGTERM ? T ? Termination signal. ?

?SIGTSTP ? S ? Terminal stop signal. ?

?SIGTTIN ? S ? Background process attempting read. ?

?SIGTTOU ? S ? Background process attempting write. ?

?SIGUSR1 ? T ? User-defined signal 1. ?

?SIGUSR2 ? T ? User-defined signal 2. ?

?SIGPOLL ? T ? Pollable event. ?

?SIGPROF ? T ? Profiling timer expired. ?

?SIGSYS ? A ? Bad system call. ?

?SIGTRAP ? A ? Trace/breakpoint trap. ?

?SIGURG ? I ? High bandwidth data is available at a socket. ?

?SIGVTALRM ? T ? Virtual timer expired. ?

?SIGXCPU ? A ? CPU time limit exceeded. ?

?SIGXFSZ ? A ? File size limit exceeded. ?

? ? ? ?

??

The default actions are as follows:

T Abnormal termination of the process.

A Abnormal termination of the process with additional actions.

I Ignore the signal.

S Stop the process.

C Continue the process, if it is stopped; otherwise, ignore the signal.

The effects on the process in each case are described in the System In? terfaces volume of POSIX.1?2017, Section 2.4.3, Signal Actions.

The <signal.h> header shall declare the sigaction structure, which shall include at least the following members:

void (*sa_handler)(int) Pointer to a signal-catching function or one of the SIG_IGN or SIG_DFL.

sigset_t sa_mask Set of signals to be blocked during execution of the signal handling function.

int sa_flags Special flags.

void (*sa_sigaction)(int, siginfo_t *, void *) Pointer to a signal-catching function.

The storage occupied by sa_handler and sa_sigaction may overlap, and a conforming application shall not use both simultaneously.

The <signal.h> header shall define the following macros which shall ex? pand to integer constant expressions that need not be usable in #if preprocessing directives:

SIG_BLOCK The resulting set is the union of the current set and the signal set pointed to by the argument set.

SIG_UNBLOCK The resulting set is the intersection of the current set and the complement of the signal set pointed to by the argument set.

SIG_SETMASK The resulting set is the signal set pointed to by the ar? gument set.

The <signal.h> header shall also define the following symbolic con? stants:

SA_NOCLDSTOP Do not generate SIGCHLD when children stop or stopped children continue.

SA_ONSTACK Causes signal delivery to occur on an alternate stack.

SA_RESETHAND Causes signal dispositions to be set to SIG_DFL on entry

to signal handlers.

SA_RESTART Causes certain functions to become restartable.

SA_SIGINFO Causes extra information to be passed to signal handlers at the time of receipt of a signal.

SA_NOCLDWAIT Causes implementations not to create zombie processes or status information on child termination. See sigaction().

SA_NODEFER Causes signal not to be automatically blocked on entry to signal handler.

SS_ONSTACK Process is executing on an alternate signal stack.

SS_DISABLE Alternate signal stack is disabled.

MINSIGSTKSZ Minimum stack size for a signal handler.

SIGSTKSZ Default size in bytes for the alternate signal stack.

The <signal.h> header shall define the mcontext_t type through typedef.

The <signal.h> header shall define the ucontext_t type as a structure that shall include at least the following members:

ucontext_t *uc_link Pointer to the context that is resumed when this context returns.

sigset_t uc_sigmask The set of signals that are blocked when this context is active.

stack_t uc_stack The stack used by this context.

mcontext_t uc_mcontext A machine-specific representation of the saved context.

The <signal.h> header shall define the stack_t type as a structure, which shall include at least the following members:

void *ss_sp Stack base or pointer.

size_t ss_size Stack size.

int ss_flags Flags.

The <signal.h> header shall define the siginfo_t type as a structure, which shall include at least the following members:

int si_signo Signal number.

int si_code Signal code.

int si_errno If non-zero, an errno value associated with this signal, as described in <errno.h>.

pid_t si_pid Sending process ID.

uid_t si_uid Real user ID of sending process.

void *si_addr Address of faulting instruction.

int si_status Exit value or signal.

long si_band Band event for SIGPOLL.

union sigval si_value Signal value.

The <signal.h> header shall define the symbolic constants in the Code column of the following table for use as values of si_code that are signal-specific or non-signal-specific reasons why the signal was gen? erated.

??

?Signal ? Code ? Reason ?

??

?SIGILL ?ILL_ILLOPC ?Illegal opcode. ?

? ?ILL_ILLOPN ?Illegal operand. ?

? ?ILL_ILLADR ?Illegal addressing mode. ?

? ?ILL_ILLTRP ?Illegal trap. ?

? ?ILL_PRVOPC ?Privileged opcode. ?

? ?ILL_PRVREG ?Privileged register. ?

? ?ILL_COPROC ?Coprocessor error. ?

? ?ILL_BADSTK ?Internal stack error. ?

??

?SIGFPE ?FPE_INTDIV ?Integer divide by zero. ?

? ?FPE_INTOVF ?Integer overflow. ?

? ?FPE_FLTDIV ?Floating-point divide by zero. ?

? ?FPE_FLTOVF ?Floating-point overflow. ?

? ?FPE_FLTUND ?Floating-point underflow. ?

? ?FPE_FLTRES ?Floating-point inexact result. ?

? ?FPE_FLTINV ?Invalid floating-point operation. ?

? ?FPE_FLTSUB ?Subscript out of range. ?

??

?SIGSEGV?SEGV_MAPERR ?Address not mapped to object. ?

? ?SEGV_ACCERR ?Invalid permissions for mapped object. ?

??

?SIGBUS ?BUS_ADRALN ?Invalid address alignment. ?

? ?BUS_ADRERR ?Nonexistent physical address. ?

? ?BUS_OBJERR ?Object-specific hardware error. ?

??

?SIGTRAP?TRAP_BRKPT ?Process breakpoint. ?

? ?TRAP_TRACE ?Process trace trap. ?

??

?SIGCHLD?CLD_EXITED ?Child has exited. ?

? ?CLD_KILLED ?Child has terminated abnormally and did not create a core file. ?

? ?CLD_DUMPED ?Child has terminated abnormally and created a core file. ?

? ?CLD_TRAPPED ?Traced child has trapped. ?

? ?CLD_STOPPED ?Child has stopped. ?

? ?CLD_CONTINUED?Stopped child has continued. ?

??

?SIGPOLL?POLL_IN ?Data input available. ?

? ?POLL_OUT ?Output buffers available. ?

? ?POLL_MSG ?Input message available. ?

? ?POLL_ERR ?I/O error. ?

? ?POLL_PRI ?High priority input available. ?

? ?POLL_HUP ?Device disconnected. ?

??

?Any ?SI_USER ?Signal sent by kill(). ?

? ?SI_QUEUE ?Signal sent by sigqueue(). ?

? ?SI_TIMER ?Signal generated by expiration of a timer set by timer_settime(). ?

? ?SI_ASYNCIO ?Signal generated by completion of an asynchronous I/O ?

? ? ?request. ?

? ?SI_MESGQ ?Signal generated by arrival of a message on an empty message ?

? ? ?queue. ?

??

Implementations may support additional si_code values not included in this list, may generate values included in this list under circum?stances other than those described in this list, and may contain exten?

sions or limitations that prevent some values from being generated. Im?
plementations do not generate a different value from the ones described
in this list for circumstances described in this list.
In addition, the following signal-specific information shall be avail?
able:

```
????????????????????????????????????????????????????????????????????????????????
?Signal  ?     Member     ?                  Value                    ?
????????????????????????????????????????????????????????????????????????????????
?SIGILL  ? void * si_addr ? Address of faulting instruction.               ?
?SIGFPE  ?                ?                                  ?
????????????????????????????????????????????????????????????????????????????????
?SIGSEGV ? void * si_addr ? Address of faulting memory reference.          ?
?SIGBUS  ?                ?                                  ?
????????????????????????????????????????????????????????????????????????????????
?SIGCHLD ? pid_t si_pid   ? Child process ID.                              ?
?        ? int si_status  ? If si_code is equal to CLD_EXITED, then si_status ?
?        ?                ? holds  the  exit value of the process; otherwise, ?
?        ?                ? it is equal to the signal that caused the process ?
?        ?                ? to  change  state.  The  exit  value in si_status ?
?        ?                ? shall be equal to the full exit value  (that  is, ?
?        ?                ? the  value passed to _exit(), _Exit(), or exit(), ?
?        ?                ? or returned from main()); it shall not be limited ?
?        ?                ? to the least significant eight bits of the value. ?
?        ? uid_t si_uid   ? Real user ID of the process that sent the signal. ?
????????????????????????????????????????????????????????????????????????????????
?SIGPOLL ? long si_band   ? Band event for POLL_IN, POLL_OUT, or POLL_MSG.    ?
????????????????????????????????????????????????????????????????????????????????
```

For some implementations, the value of si_addr may be inaccurate.
The following shall be declared as functions and may also be defined as
macros. Function prototypes shall be provided.

```
int    kill(pid_t, int);
int    killpg(pid_t, int);
void   psiginfo(const siginfo_t *, const char *);
```

```
void   psignal(int, const char *);
int    pthread_kill(pthread_t, int);
int    pthread_sigmask(int, const sigset_t *restrict,
           sigset_t *restrict);
int    raise(int);
int    sigaction(int, const struct sigaction *restrict,
           struct sigaction *restrict);
int    sigaddset(sigset_t *, int);
int    sigaltstack(const stack_t *restrict, stack_t *restrict);
int    sigdelset(sigset_t *, int);
int    sigemptyset(sigset_t *);
int    sigfillset(sigset_t *);
int    sighold(int);
int    sigignore(int);
int    siginterrupt(int, int);
int    sigismember(const sigset_t *, int);
void (*signal(int, void (*)(int)))(int);
int    sigpause(int);
int    sigpending(sigset_t *);
int    sigprocmask(int, const sigset_t *restrict, sigset_t *restrict);
int    sigqueue(pid_t, int, union sigval);
int    sigrelse(int);
void (*sigset(int, void (*)(int)))(int);
int    sigsuspend(const sigset_t *);
int    sigtimedwait(const sigset_t *restrict, siginfo_t *restrict,
           const struct timespec *restrict);
int    sigwait(const sigset_t *restrict, int *restrict);
int    sigwaitinfo(const sigset_t *restrict, siginfo_t *restrict);
```

Inclusion of the <signal.h> header may make visible all symbols from the <time.h> header.

The following sections are informative.

APPLICATION USAGE

On systems not supporting the XSI option, the si_pid and si_uid members

of siginfo_t are only required to be valid when si_code is SI_USER or SI_QUEUE. On XSI-conforming systems, they are also valid for all si_code values less than or equal to 0; however, it is unspecified whether SI_USER and SI_QUEUE have values less than or equal to zero, and therefore XSI applications should check whether si_code has the value SI_USER or SI_QUEUE or is less than or equal to 0 to tell whether si_pid and si_uid are valid.

RATIONALE

None.

FUTURE DIRECTIONS

The SIGPOLL and SIGPROF signals may be removed in a future version.

SEE ALSO

<errno.h>, <stropts.h>, <sys_types.h>, <time.h>

The System Interfaces volume of POSIX.1?2017, Section 2.2, The Compila? tion Environment, alarm(), ioctl(), kill(), killpg(), psiginfo(), pthread_kill(), pthread_sigmask(), raise(), sigaction(), sigaddset(), sigaltstack(), sigdelset(), sigemptyset(), sigfillset(), sighold(), siginterrupt(), sigismember(), signal(), sigpending(), sigqueue(), sig? suspend(), sigtimedwait(), sigwait(), timer_create(), wait(), waitid()

The Shell and Utilities volume of POSIX.1?2017, kill

COPYRIGHT

nel.org/doc/man-pages/reporting_bugs.html .