NAME signal – base signals

SYNOPSIS **#include <signal.h>**

DESCRIPTION A signal is an asynchronous notification of an event. A signal is said to be generated for (or sent to) a process when the event associated with that signal first occurs. Examples of such events include hardware faults, timer expiration and terminal activity, as well as the invocation of the **kill**(2) or **sigsend**(2) system calls. In some circumstances, the same event generates signals for multiple processes. A process may request a detailed notification of the source of the signal and the reason why it was generated (see **siginfo**(5)).

A process responds to signals in similar ways whether it is using threads (see **thr_create**(3T)) or it is using lightweight processes (LWPs). Each process may specify a system action to be taken in response to each signal sent to it, called the signal's disposition. All threads or LWPs in the process share the disposition. The set of system signal actions for a process is initialized from that of its parent. Once an action is installed for a specific signal, it usually remains installed until another disposition is explicitly requested by a call to either **sigaction, signal** or **sigset,** or until the process **execs** (see **sigaction**(2) and **signal**(3C)). When a process execs, all signals whose disposition has been set to catch the signal will be set to **SIG_DFL.** Alternatively, a process may request that the system automatically reset the disposition of a signal to **SIG_DFL** after it has been caught (see **sigaction**(2) and **signal**(3C)).

A signal is said to be delivered to a process when a thread or LWP within the process takes the appropriate action for the disposition and signal. Delivery of a signal can be blocked. Each thread or LWP has a signal mask (see **thr_sigsetmask**(3T) or **sigprocmask**(2)) that defines the set of signals currently blocked from delivery to it. The signal mask of the main thread or LWP is inherited from the signal mask of the thread or LWP that created it in the parent process. The selection of the thread or LWP within the process that is to take the appropriate action for the signal is based on the method of signal generation and the signal masks of the threads or LWPs in the receiving process. Signals that are a generated by action of a particular thread or LWP such as hardware faults are delivered to the thread or LWP that caused the signal. Also, see **alarm**(2) for current semantics of delivery of **SIGALRM.** Signals that are directed to a particular thread or LWP (see **thr_kill**(3T) or **_lwp_kill**(2)) are delivered to the targeted thread or LWP. If the selected thread or LWP has blocked the signal, it remains pending on the thread or LWP until it is unblocked. For all other types of signal generation (e.g. **kill**(2), **sigsend**(2), terminal activity, and other external events not ascribable to a particular thread or LWP) one of the threads or LWPs that does not have the signal blocked is selected to process the signal. If all the threads or LWPs within the process block the signal, it remains pending on the process until a thread or LWP in the process unblocks it. If the action associated with a signal is set to ignore the signal then both currently pending and subsequently generated signals of this type are discarded immediately for this process.

The determination of which action is taken in response to a signal is made at the time the signal is delivered to a thread or LWP within the process, allowing for any changes since the time of generation. This determination is independent of the means by which the signal was originally generated.

The signals currently defined by **<signal.h>** are as follows:

Name	Value	Default	Event
SIGHUP	1	Exit	Hangup (see **termio**(7I))
SIGINT	2	Exit	Interrupt (see **termio**(7I))
SIGQUIT	3	Core	Quit (see **termio**(7I))
SIGILL	4	Core	Illegal Instruction
SIGTRAP	5	Core	Trace/Breakpoint Trap
SIGABRT	6	Core	Abort
SIGEMT	7	Core	Emulation Trap
SIGFPE	8	Core	Arithmetic Exception
SIGKILL	9	Exit	Killed
SIGBUS	10	Core	Bus Error
SIGSEGV	11	Core	Segmentation Fault
SIGSYS	12	Core	Bad System Call
SIGPIPE	13	Exit	Broken Pipe
SIGALRM	14	Exit	Alarm Clock
SIGTERM	15	Exit	Terminated
SIGUSR1	16	Exit	User Signal 1
SIGUSR2	17	Exit	User Signal 2
SIGCHLD	18	Ignore	Child Status Changed
SIGPWR	19	Ignore	Power Fail/Restart
SIGWINCH	20	Ignore	Window Size Change
SIGURG	21	Ignore	Urgent Socket Condition
SIGPOLL	22	Exit	Pollable Event (see **streamio**(7I))
SIGSTOP	23	Stop	Stopped (signal)
SIGTSTP	24	Stop	Stopped (user) (see **termio**(7I))
SIGCONT	25	Ignore	Continued
SIGTTIN	26	Stop	Stopped (tty input) (see **termio**(7I))
SIGTTOU	27	Stop	Stopped (tty output) (see **termio**(7I))
SIGVTALRM	28	Exit	Virtual Timer Expired
SIGPROF	29	Exit	Profiling Timer Expired
SIGXCPU	30	Core	CPU time limit exceeded (see **getrlimit**(2))
SIGXFSZ	31	Core	File size limit exceeded (see **getrlimit**(2))
SIGWAITING	32	Ignore	Concurrency signal reserved by threads library
SIGLWP	33	Ignore	Inter-LWP signal reserved by threads library
SIGFREEZE	34	Ignore	Check point Freeze
SIGTHAW	35	Ignore	Check point Thaw
SIGCANCEL	36	Ignore	Cancellation signal reserved by threads library
SIGRTMIN	*	Exit	First real time signal
(SIGRTMIN + 1)	*	Exit	Second real time signal
. . .			
(SIGRTMAX - 1)	*	Exit	Second-to-last real time signal
SIGRTMAX	*	Exit	Last real time signal

(The symbols **SIGRTMIN** through **SIGRTMAX** are evaluated dynamically in order to permit future configurability)

A process, using a **signal**(3C), **sigset**(3C) or **sigaction**(2) system call, may specify one of three dispositions for a signal: take the default action for the signal, ignore the signal, or catch the signal.

Default Action: SIG_DFL

A disposition of **SIG_DFL** specifies the default action. The default action for each signal is listed in the table above and is selected from the following:

Exit When it gets the signal, the receiving process is to be terminated with all the consequences outlined in **exit**(2).

Core When it gets the signal, the receiving process is to be terminated with all the consequences outlined in **exit**(2). In addition, a ''core image'' of the process is constructed in the current working directory.

Stop When it gets the signal, the receiving process is to stop. When a process is stopped, all the threads and LWPs within the process also stop executing.

Ignore When it gets the signal, the receiving process is to ignore it. This is identical to setting the disposition to **SIG_IGN.**

Ignore Signal: SIG_IGN

A disposition of **SIG_IGN** specifies that the signal is to be ignored. Setting a signal action to **SIG_IGN** for a signal that is pending causes the pending signal to be discarded, whether or not it is blocked. Any queued values pending are also discarded, and the resources used to queue them are released and made available to queue other signals.

Catch Signal: *function address*

A disposition that is a function address specifies that, when it gets the signal, the thread or LWP within the process that is selected to process the signal will execute the signal handler at the specified address. Normally, the signal handler is passed the signal number as its only argument; if the disposition was set with the **sigaction** function however, additional arguments may be requested (see **sigaction**(2)). When the signal handler returns, the receiving process resumes execution at the point it was interrupted, unless the signal handler makes other arrangements. If an invalid function address is specified, results are undefined.

If the disposition has been set with the **sigset** or **sigaction** function, the signal is automatically blocked in the thread or LWP while it is executing the signal catcher. If a **longjmp** (see **setjmp**(3C)) is used to leave the signal catcher, then the signal must be explicitly unblocked by the user (see **signal**(3C) and **sigprocmask**(2)).

If execution of the signal handler interrupts a blocked system call, the handler is executed and the interrupted system call returns a –1 to the calling process with **errno** set to **EINTR.** However, if the **SA_RESTART** flag is set the system call will be transparently restarted.

Some signal–generating functions, such as high resolution timer expiration, asynchronous I/O completion, inter-process message arrival, and the **sigqueue**(3R) function, support the specification of an application defined value, either explicitly as a parameter to the function, or in a **sigevent** structure parameter. The **sigevent** structure is defined by **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	**sigev_notify**	Notification type
int	**sigev_signo**	Signal number
union sigval	**sigev_value**	Signal value

The **sigval** union is defined by **<signal.h>** and contains at least the following members:

Member Type	Member Name	Description
int	**sival_int**	Integer signal value
void *	**sival_ptr**	Pointer signal value

sigev_notify specifies the notification mechanism to use when an asynchronous event occurs. **sigev_notify** may be defined with the following values:

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

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

Your implementation may define additional notification mechanisms.

sigev_signo specifies the signal to be generated.

sigev_value references the application defined value to be passed to the signal-catching function at the time of the signal delivery as the **si_value** member of the **siginfo_t** structure.

The **sival_int** member will be used when the application defined value is of type **int**; and the **sival_ptr** member will be used when the application defined value is a pointer.

When a signal is generated by **sigqueue**(3R) or any signal–generating function which supports the specification of an application defined value, the signal is marked pending and, if the **SA_SIGINFO** flag is set for that signal, the signal is queued to the process along with the application specified signal value. Multiple occurrences of signals so generated are queued in FIFO order. If the **SA_SIGINFO** flag is not set for that signal, later occurrences of that signal's generation, when a signal is already queued, are silently discarded.

SEE ALSO

_lwp_sigredirect(2), **alarm**(2), **intro**(2), **exit**(2), **getrlimit**(2), **kill**(2), **pause**(2), **sigaction**(2), **sigaltstack**(2), **sigprocmask**(2), **sigsend**(2), **sigsuspend**(2), **wait**(2), **signal**(3C), **sigsetops**(3C), **sigqueue**(3R), **siginfo**(5), **ucontext**(5)

NOTES

The dispositions of the **SIGKILL** and **SIGSTOP** signals cannot be altered from their default values. The system generates an error if this is attempted.

The **SIGKILL** and **SIGSTOP** signals cannot be blocked. The system silently enforces this restriction.

Whenever a process receives a **SIGSTOP, SIGTSTP, SIGTTIN,** or **SIGTTOU** signal, regardless of its disposition, any pending **SIGCONT** signal are discarded.

Whenever a process receives a **SIGCONT** signal, regardless of its disposition, any pending **SIGSTOP, SIGTSTP, SIGTTIN,** and **SIGTTOU** signals is discarded. In addition, if the process was stopped, it is continued.

SIGPOLL is issued when a file descriptor corresponding to a STREAMS (see **intro**(2)) file has a ''selectable'' event pending. A process must specifically request that this signal be sent using the **I_SETSIG ioctl** call. Otherwise, the process will never receive **SIGPOLL.**

If the disposition of the **SIGCHLD** signal has been set with **signal** or **sigset**, or with **sigaction** and the **SA_NOCLDSTOP** flag has been specified, it will only be sent to the calling process when its children exit; otherwise, it will also be sent when the calling process's children are stopped or continued due to job control.

The name **SIGCLD** is also defined in this header and identifies the same signal as **SIGCHLD. SIGCLD** is provided for backward compatibility, new applications should use **SIGCHLD.**

The disposition of signals that are inherited as **SIG_IGN** should not be changed.

A signal directed via **kill**(2), **sigqueue**(2), **sigsend**(2), terminal activity, and other external events not ascribable to a particular thread or LWP, such as the **SIGXFSZ** or **SIGPIPE** signal, to a multi-threaded process, i.e. a process linked with **-lthread** or **-lpthread**, is routed to this process through a special, designated LWP within this process, called the **aslwp** (short for "Asynchronous Signal LWP").

The **aslwp** within the multi-threaded process receives notification of any signal directed to this process. Upon receiving this notification, the **aslwp** forwards it to a thread within the process that has the signal unmasked. Actual signal delivery to the thread occurs only when the thread is running on an LWP. If no threads exist having that signal number unblocked, the signal remains pending. The **aslwp** is usually blocked in a call to **_signotifywait**(2), waiting for such notifications. The eventual target thread receives the signal via a call to **_lwp_sigredirect**(2), made either by the **aslwp** or the thread itself, redirecting the signal to the LWP that the target thread is running on.