NAME

siginfo - signal generation information

SYNOPSIS

#include <signal.h>

DESCRIPTION

A process may request signal information when it is catching a signal. The information specifies why the system generated that signal. To request signal information in a signal handler, the user can set SA_SIGINFO in *sa_flags* before sigaction(2) is called, otherwise the user can use sigwaitinfo(2) and sigtimedwait(2) to get signal information. In either case, the system returns the information in a structure of type *siginfo_t*, which includes the following information:

Type	**Member**	**Description**
int	*si_signo*	signal number
int	*si_errno*	error number
int	*si_code*	signal code
union sigval	*si_value*	signal value
pid_t	*si_pid*	sending process ID
uid_t	*si_uid*	sending process’s real user ID
void	**si_addr*	virtual address
int	*si_status*	exit value or signal
long	*si_band*	band event for SIGPOLL
int	*si_trapno*	machine trap code
int	*si_timerid*	POSIX timer ID
int	*si_overrun*	POSIX timer overrun count
int	*si_mqd*	POSIX message queue ID
int	*si_syscall*	system-call number for system calls blocked by Capsicum

The *si_signo* member contains the signal number.

The *si_errno* member contains an error number defined in the file <*errno.h*>.

The *si_code* member contains a code which describes the cause of the signal. The macros specified in the **Code** column of the following table are defined for use as values of *si_code* that are signal-specific or non-signal-specific reasons why the signal was generated:

Signal	**Code**	**Reason**
SIGILL	ILL_ILLOPC	illegal opcode
	ILL_ILLOPN	illegal operand

	ILL_ILLADR	illegal addressing mode
	ILL_ILLTRP	illegal trap
	ILL_PRVOPC	illegal privileged opcode
	ILL_PRVREG	illegal 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
	BUS_OOMERR	cannot alloc a page to map at fault
SIGTRAP	TRAP_BRKPT	process breakpoint
	TRAP_TRACE	process trace trap
	TRAP_DTRACE	DTrace induced trap
	TRAP_CAP	capabilities protective 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_NOINFO	Only the *si_signo* member is meaningful; the value of all other members is unspecified.
	SI_USER	signal sent by kill(2)

SI_QUEUE	signal sent by sigqueue(2)
SI_TIMER	signal generated by expiration of a timer set by timer_settime(2)
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
SI_KERNEL	signal generated by miscellaneous parts of the kernel
SI_LWP	signal sent by pthread_kill(3)

For synchronous signals, *si_addr* is generally set to the address of the faulting instruction. However, synchronous signals raised by a faulting memory access such as SIGSEGV and SIGBUS may report the address of the faulting memory access (if available) in *si_addr* instead. Additionally SIGTRAP raised by a hardware watchpoint exception may report the data address that triggered the watchpoint in *si_addr*.

Sychronous signals set *si_trapno* to a machine-dependent trap number.

In addition, the following signal-specific information is available:

Signal	**Member**	**Value**
SIGCHLD	*si_pid*	child process ID
	si_status	exit value or signal; if *si_code* is equal to CLD_EXITED, then it is equal to the exit value of the child process, otherwise, it is equal to a signal that caused the child process to change state.
	si_uid	real user ID of the process that sent the signal
SIGPOLL	*si_band*	band event for POLL_IN, POLL_OUT, or POLL_MSG

Finally, the following code-specific information is available:

Code	**Member**	**Value**
SI_USER	*si_pid*	the process ID that sent the signal
	si_uid	real user ID of the process that sent the signal
SI_QUEUE	*si_value*	the value passed to sigqueue(2) system call
	si_pid	the process ID that sent the signal
	si_uid	real user ID of the process that sent the signal
SI_TIMER	*si_value*	the value passed to timer_create(2) system call
	si_timerid	the timer ID returned by timer_create(2) system call
	si_overrun	timer overrun count corresponding to the signal
	si_errno	If timer overrun will be {DELAYTIMER_MAX}, an error code

		defined in <*errno.h*> is set
SI_ASYNCIO	*si_value*	the value passed to aio system calls
SI_MESGQ	*si_value*	the value passed to mq_notify(2) system call
	si_mqd	the ID of the message queue which generated the signal
SI_LWP	*si_pid*	the process ID that sent the signal
	si_uid	real user ID of the process that sent the signal

NOTES

Currently, the kernel never generates the SIGPOLL signal. SIGCHLD signal is queued when a process changed its status or exited. POSIX Realtime Extensions like aio, timer, and message queue also queue signals. Signals with code SI_USER, SI_KERNEL or SI_LWP are only queued if there are sufficient resources; otherwise, SI_NOINFO results. For some hardware architectures, the exact value of *si_addr* might not be available.

SEE ALSO

aio_read(2), kill(2), mq_notify(2), sigaction(2), sigqueue(2), sigwaitinfo(2), timer_create(2), timer_settime(2), waitpid(2), pthread_kill(3)

STANDARDS

The *siginfo_t* type conforms to IEEE Std 1003.1-2004 ("POSIX.1").

HISTORY

Full support for POSIX signal information first appeared in FreeBSD 7.0. The codes SI_USER and SI_KERNEL can be generated as of FreeBSD 8.1. The code SI_LWP can be generated as of FreeBSD 9.0.

AUTHORS

This manual page was written by David Xu <*davidxu@FreeBSD.org*>.