NAME

signal, **SIGADDSET**, **SIGDELSET**, **SETEMPTYSET**, **SIGFILLSET**, **SIGISMEMBER**, **SIGISEMPTY**, **SIGNOTEMPTY**, **SIGSETEQ**, **SIGSETNEQ**, **SIGSETOR**, **SIGSETAND**, **SIGSETNAND**, **SIGSETCANTMASK**, **SIG_STOPSIGMASK**, **SIG_CONTSIGMASK**, **SIGPENDING**, **cursig**, **execsigs**, **issignal**, **killproc**, **pgsigio**, **postsig**, **sigexit**, **siginit**, **signotify**, **trapsignal** - kernel signal functions

SYNOPSIS

#include <sys/param.h>
#include <sys/proc.h>
#include <sys/signalvar.h>

void
SIGADDSET(*sigset_t set, int signo*);

void
SIGDELSET(*sigset_t set, int signo*);

void
SIGEMPTYSET(*sigset_t set*);

void
SIGFILLSET(*sigset_t set*);

int
SIGISMEMBER(*sigset_t set, int signo*);

int
SIGISEMPTY(*sigset_t set*);

int
SIGNOTEMPTY(*sigset_t set*);

int
SIGSETEQ(*sigset_t set1, sigset_t set2*);

int
SIGSETNEQ(*sigset_t set1, sigset_t set2*);

void
SIGSETOR(*sigset_t set1, sigset_t set2*);

void
SIGSETAND(*sigset_t set1*, *sigset_t set2*);

void
SIGSETNAND(*sigset_t set1*, *sigset_t set2*);

void
SIG_CANTMASK(*sigset_t set*);

void
SIG_STOPSIGMASK(*sigset_t set*);

void
SIG_CONTSIGMASK(*sigset_t set*);

int
SIGPENDING(*struct proc *p*);

int
cursig(*struct thread *td*);

void
execsigs(*struct proc *p*);

int
issignal(*struct thread *td*);

void
killproc(*struct proc *p*, *char *why*);

void
pgsigio(*struct sigio **sigiop*, *int sig*, *int checkctty*);

void
postsig(*int sig*);

void
sigexit(*struct thread *td*, *int signum*);

void

siginit(*struct proc *p*);

void
signotify(*struct thread *td*);

void
trapsignal(*struct thread *td, int sig, u_long code*);

DESCRIPTION

The **SIGADDSET**() macro adds *signo* to *set*. No effort is made to ensure that *signo* is a valid signal number.

The **SIGDELSET**() macro removes *signo* from *set*. No effort is made to ensure that *signo* is a valid signal number.

The **SIGEMPTYSET**() macro clears all signals in *set*.

The **SIGFILLSET**() macro sets all signals in *set*.

The **SIGISMEMBER**() macro determines if *signo* is set in *set*.

The **SIGISEMPTY**() macro determines if *set* does not have any signals set.

The **SIGNOTEMPTY**() macro determines if *set* has any signals set.

The **SIGSETEQ**() macro determines if two signal sets are equal; that is, the same signals are set in both.

The **SIGSETNEQ**() macro determines if two signal sets differ; that is, if any signal set in one is not set in the other.

The **SIGSETOR**() macro ORs the signals set in *set2* into *set1*.

The **SIGSETAND**() macro ANDs the signals set in *set2* into *set1*.

The **SIGSETNAND**() macro NANDs the signals set in *set2* into *set1*.

The **SIG_CANTMASK**() macro clears the SIGKILL and SIGSTOP signals from *set*. These two signals cannot be blocked or caught and **SIG_CANTMASK**() is used in code where signals are manipulated to ensure this policy is enforced.

The **SIG_STOPSIGMASK**() macro clears the SIGSTOP, SIGTSTP, SIGTTIN, and SIGTTOU signals from *set*. **SIG_STOPSIGMASK**() is used to clear stop signals when a process is waiting for a child to exit or exec, and when a process is continuing after having been suspended.

The **SIG_CONTSIGMASK**() macro clears the SIGCONT signal from *set*. **SIG_CONTSIGMASK**() is called when a process is stopped.

The **SIGPENDING**() macro determines if the given process has any pending signals that are not masked. If the process has a pending signal and the process is currently being traced, **SIGPENDING**() will return true even if the signal is masked.

The **cursig**() function returns the signal number that should be delivered to process *td->td_proc*. If there are no signals pending, zero is returned.

The **execsigs**() function resets the signal set and signal stack of a process in preparation for an execve(2). The process lock for *p* must be held before **execsigs**() is called.

The **issignal**() function determines if there are any pending signals for process *td->td_proc* that should be caught, or cause this process to terminate or interrupt its current system call. If process *td->td_proc* is currently being traced, ignored signals will be handled and the process is always stopped. Stop signals are handled and cleared right away by **issignal**() unless the process is a member of an orphaned process group and the stop signal originated from a TTY. The process spin lock for *td->td_proc* may be acquired and released. The *sigacts* structure *td->td_proc->p_sigacts* must be locked before calling **issignal**() and may be released and reacquired during the call. The process lock for *td->td_proc* must be acquired before calling **issignal**() and may be released and reacquired during the call. Default signal actions are not taken for system processes and init.

The **killproc**() function delivers SIGKILL to *p*. *why* is logged as the reason *why* the process was killed.

The **pgsigio**() function sends the signal *sig* to the process or process group *sigiop->sio_pgid*. If *checkctty* is non-zero, the signal is only delivered to processes in the process group that have a controlling terminal. If *sigiop->sio_pgid* is for a process (> 0), the lock for *sigiop->sio_proc* is acquired and released. If *sigiop->sio_pgid* is for a process group (< 0), the process group lock for *sigiop->sio_pgrp* is acquired and released. The lock *sigio_lock* is acquired and released.

The **postsig**() function handles the actual delivery of the signal *sig*. **postsig**() is called from **ast**() after the kernel has been notified that a signal should be delivered (via a call to **signotify**(), which causes the flag PS_NEEDSIGCHK to be set). The process lock for process that owns *curthread* must be held before **postsig**() is called, and the current process cannot be 0. The lock for the *p_sigacts* field of the current process must be held before **postsig**() is called, and may be released and reacquired.

The **sigexit**() function causes the process that owns *td* to exit with a return value of signal number *sig*. If required, the process will dump core. The process lock for the process that owns *td* must be held before **sigexit**() is called.

The **siginit**() function is called during system initialization to cause every signal with a default property of SA_IGNORE (except SIGCONT) to be ignored by *p*. The process lock for *p* is acquired and released, as is the lock for sigacts structure *p->p_sigacts*. The only process that **siginit**() is ever called for is *proc0*.

The **signotify**() function flags that there are unmasked signals pending that **ast**() should handle. The process lock for process *td->td_proc* must be held before **signotify**() is called, and the thread lock is acquired and released.

The **trapsignal**() function sends a signal that is the result of a trap to process *td->td_proc*. If the process is not being traced and the signal can be delivered immediately, **trapsignal**() will deliver it directly; otherwise, **trapsignal**() will call psignal(9) to cause the signal to be delivered. The process lock for *td->td_proc* is acquired and released. The lock for the *p_sigacts* field of *td->td_proc* is acquired and released.

RETURN VALUES

The **SIGISMEMBER**(), **SIGISEMPTY**(), **SIGNOTEMPTY**(), **SIGSETEQ**(), **SIGSETNEQ**(), and **SIGPENDING**() macros all return non-zero (true) if the condition they are checking is found to be true; otherwise, zero (false) is returned.

The **cursig**() function returns either a valid signal number or zero.

issignal() returns either a valid signal number or zero.

SEE ALSO

pgsignal(9), psignal(9)

AUTHORS

This manual page was written by Chad David <*davidc@FreeBSD.org*>.