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 'pthread_sigmask.3p' command

$ man pthread_sigmask.3p

PTHREAD_SIGMASK(3P) POSIX Programmer's Manual PTHREAD_SIGMASK(3P)

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

pthread_sigmask, sigprocmask ? examine and change blocked signals

SYNOPSIS

```
#include <signal.h>
int pthread_sigmask(int how, const sigset_t *restrict set,
    sigset_t *restrict oset);
int sigprocmask(int how, const sigset_t *restrict set,
    sigset_t *restrict oset);
```

DESCRIPTION

The pthread_sigmask() function shall examine or change (or both) the calling thread's signal mask, regardless of the number of threads in the process. The function shall be equivalent to sigprocmask(), without the restriction that the call be made in a single-threaded process.

In a single-threaded process, the sigprocmask() function shall examine or change (or both) the signal mask of the calling thread.

If the argument set is not a null pointer, it points to a set of sig? nals to be used to change the currently blocked set.

The argument how indicates the way in which the set is changed, and the application shall ensure it consists of one of the following values:

SIG_BLOCK The resulting set shall be the union of the current set and the signal set pointed to by set.

SIG_SETMASK The resulting set shall be the signal set pointed to by set.

SIG_UNBLOCK The resulting set shall be the intersection of the current set and the complement of the signal set pointed to by set.

If the argument oset is not a null pointer, the previous mask shall be stored in the location pointed to by oset. If set is a null pointer, the value of the argument how is not significant and the thread's signal mask shall be unchanged; thus the call can be used to enquire about currently blocked signals.

If there are any pending unblocked signals after the call to sigprocmask(), at least one of those signals shall be delivered before the call to sigprocmask() returns.

It is not possible to block those signals which cannot be ignored. This shall be enforced by the system without causing an error to be indicated.

If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked, the result is undefined, unless the signal was generated by the action of another process, or by one of the functions kill(), pthread_kill(), raise(), or sigqueue().

If sigprocmask() fails, the thread's signal mask shall not be changed.

The use of the sigprocmask() function is unspecified in a multi-threaded process.

RETURN VALUE

Upon successful completion pthread_sigmask() shall return 0; otherwise, it shall return the corresponding error number.

Upon successful completion, sigprocmask() shall return 0; otherwise, -1 shall be returned, errno shall be set to indicate the error, and the signal mask of the process shall be unchanged.

ERRORS

The pthread_sigmask() and sigprocmask() functions shall fail if:

EINVAL The value of the how argument is not equal to one of the defined values.

The pthread_sigmask() function shall not return an error code of [EINTR].

The following sections are informative.

EXAMPLES

Signaling in a Multi-Threaded Process

This example shows the use of pthread_sigmask() in order to deal with signals in a multi-threaded process. It provides a fairly general framework that could be easily adapted/extended.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <signal.h>
#include <string.h>
#include <errno.h>
...
static sigset_t   signal_mask;  /* signals to block         */
int main (int argc, char *argv[])
{
    pthread_t  sig_thr_id;      /* signal handler thread ID */
    int        rc;              /* return code              */
    sigemptyset (&signal_mask);
    sigaddset (&signal_mask, SIGINT);
    sigaddset (&signal_mask, SIGTERM);
    rc = pthread_sigmask (SIG_BLOCK, &signal_mask, NULL);
    if (rc != 0) {
        /* handle error */
        ...
    }
    /* any newly created threads inherit the signal mask */
    rc = pthread_create (&sig_thr_id, NULL, signal_thread, NULL);
```

```
        if (rc != 0) {
            /* handle error */
            ...
        }
        /* APPLICATION CODE */
        ...
    }
    void *signal_thread (void *arg)
    {
        int       sig_caught;    /* signal caught       */
        int       rc;            /* returned code       */
        rc = sigwait (&signal_mask, &sig_caught);
        if (rc != 0) {
            /* handle error */
        }
        switch (sig_caught)
        {
        case SIGINT:     /* process SIGINT  */
            ...
            break;
        case SIGTERM:    /* process SIGTERM */
            ...
            break;
        default:         /* should normally not happen */
            fprintf (stderr, "\nUnexpected signal %d\n", sig_caught);
            break;
        }
    }
```

APPLICATION USAGE

None.

RATIONALE

When a thread's signal mask is changed in a signal-catching function that is installed by sigaction(), the restoration of the signal mask on

return from the signal-catching function overrides that change (see sigaction()). If the signal-catching function was installed with sig? nal(), it is unspecified whether this occurs.

See kill() for a discussion of the requirement on delivery of signals.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, kill(), sigaction(), sigaddset(), sigdelset(), sigemptyset(), sigfillset(), sigismember(), sigpending(), sigqueue(), sigsuspend()

The Base Definitions volume of POSIX.1?2017, <signal.h>

COPYRIGHT