select(3) - Linux man page

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

pselect, select - synchronous I/O multiplexing

Synopsis

#include <sys/select.h>

int pselect(int *nfds*, **fd_set *restrict** *readfds*,
fd_set *restrict *writefds*, **fd_set *restrict** *errorfds*,
const struct timespec *restrict *timeout*,
const sigset_t *restrict *sigmask*);
int select(int *nfds*, **fd_set *restrict** *readfds*,
fd_set *restrict *writefds*, **fd_set *restrict** *errorfds*,
struct timeval *restrict *timeout*);
void FD_CLR(int *fd*, **fd_set ***fdset*);
int FD_ISSET(int *fd*, **fd_set ***fdset*);
void FD_SET(int *fd*, **fd_set ***fdset*);
void FD_ZERO(fd_set **fdset*);

Description

The *pselect*() function shall examine the file descriptor sets whose addresses are passed in the *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively.

The *select*() function shall be equivalent to the *pselect*() function, except as follows:

* For the *select*() function, the timeout period is given in seconds and microseconds in an argument of type **struct timeval**, whereas for the *pselect*() function the timeout period is given in seconds and nanoseconds in an argument of type **struct timespec**.
* The *select*() function has no *sigmask* argument; it shall behave as *pselect*() does when *sigmask* is a null pointer.
* Upon successful completion, the *select*() function may modify the object pointed to by the *timeout* argument.

The *pselect*() and *select*() functions shall support regular files, terminal and pseudo-terminal devices, STREAMS-based files, FIFOs, pipes, and sockets. The behavior of *pselect*() and *select*() on file descriptors that refer to other types of file is unspecified.

The *nfds* argument specifies the range of descriptors to be tested. The first *nfds* descriptors shall be checked in each set; that is, the descriptors from zero through *nfds*-1 in the descriptor sets shall be examined.

If the *readfds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for being ready to read, and on output indicates which file descriptors are ready to read.

If the *writefds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for being ready to write, and on output indicates which file descriptors are ready to write.

If the *errorfds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for error conditions pending, and on output indicates which file descriptors have error conditions pending.

Upon successful completion, the *pselect*() or *select*() function shall modify the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively, and shall return the total number of ready descriptors in all the output sets. For each file descriptor less than *nfds*, the corresponding bit shall be set on successful completion if it was set on input and the associated condition is true for that file descriptor.

If none of the selected descriptors are ready for the requested operation, the *pselect*() or *select*() function shall block until at least one of the requested operations becomes ready, until the *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the *pselect*() or *select*() function shall take before timing out. If the *timeout* parameter is not a null pointer, it specifies a maximum interval to wait for the selection to complete. If the specified time interval expires without any requested operation becoming ready, the function shall return. If the *timeout* parameter is a null pointer, then the call to *pselect*() or *select*() shall block indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout* parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

The use of a timeout does not affect any pending timers set up by *alarm*(), *ualarm*(), or *setitimer*().

Implementations may place limitations on the maximum timeout interval supported. All implementations shall support a maximum timeout interval of at least 31 days. If the *timeout* argument specifies a timeout interval greater than the implementation-defined maximum value, the maximum value shall be used as the actual timeout value. Implementations may also place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval shall be rounded up to the next supported value.

If *sigmask* is not a null pointer, then the *pselect*() function shall replace the signal mask of the process by the set of signals pointed to by *sigmask* before examining the descriptors, and shall restore the signal mask of the process before returning.

A descriptor shall be considered ready for reading when a call to an input function with O_NONBLOCK clear would not block, whether or not the function would transfer data successfully. (The function might return data, an end-of-file indication, or an error other than one indicating that it is blocked, and in each of these cases the descriptor shall be considered ready for reading.)

A descriptor shall be considered ready for writing when a call to an output function with O_NONBLOCK clear would not block, whether or not the function would transfer data successfully.

If a socket has a pending error, it shall be considered to have an exceptional condition pending. Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for use with a socket, it is protocol-specific except as noted below. For other file types it is implementation-defined. If the operation is meaningless for a particular file type, *pselect*() or *select*() shall indicate that the descriptor is ready for read or write operations, and shall indicate that the descriptor has no exceptional condition pending.

If a descriptor refers to a socket, the implied input function is the *recvmsg*() function with parameters requesting normal and ancillary data, such that the presence of either type shall cause the socket to be marked as readable. The presence of out-of-band data shall be checked if the socket option SO_OOBINLINE has been enabled, as out-of-band data is enqueued with normal data. If the socket is currently listening, then it shall be marked as readable if an incoming connection request has been received, and a call to the *accept*() function shall complete without blocking.

If a descriptor refers to a socket, the implied output function is the *sendmsg*() function supplying an amount of normal data equal to the current value of the SO_SNDLOWAT option for the socket. If a non-blocking call to the *connect*() function has been made for a socket, and the connection attempt has either succeeded or failed leaving a pending error, the socket shall be marked as writable.

A socket shall be considered to have an exceptional condition pending if a receive operation with O_NONBLOCK clear for the open file description and with the MSG_OOB flag set would return out-of-band data without blocking. (It is protocol-specific whether the MSG_OOB flag would be used to read out-of-band data.) A socket shall also be considered to have an exceptional condition pending if an out-of-band data mark is present in the receive queue. Other circumstances under which a socket may be considered to have an exceptional condition pending are protocol-specific and implementation-defined.

If the *readfds*, *writefds*, and *errorfds* arguments are all null pointers and the *timeout* argument is not a null pointer, the *pselect*() or *select*() function shall block for the time specified, or until interrupted by a signal. If the *readfds*, *writefds*, and *errorfds*

arguments are all null pointers and the *timeout* argument is a null pointer, the *pselect*() or *select*() function shall block until interrupted by a signal.

File descriptors associated with regular files shall always select true for ready to read, ready to write, and error conditions.

On failure, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments shall not be modified. If the timeout interval expires without the specified condition being true for any of the specified file descriptors, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments shall have all bits set to 0.

File descriptor masks of type **fd_set** can be initialized and tested with *FD_CLR*(), *FD_ISSET*(), *FD_SET*(), and *FD_ZERO*(). It is unspecified whether each of these is a macro or a function. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with any of these names, the behavior is undefined.

FD_CLR(*fd*, *fdsetp*) shall remove the file descriptor *fd* from the set pointed to by *fdsetp*. If *fd* is not a member of this set, there shall be no effect on the set, nor will an error be returned.

FD_ISSET(*fd*, *fdsetp*) shall evaluate to non-zero if the file descriptor *fd* is a member of the set pointed to by *fdsetp*, and shall evaluate to zero otherwise.

FD_SET(*fd*, *fdsetp*) shall add the file descriptor *fd* to the set pointed to by *fdsetp*. If the file descriptor *fd* is already in this set, there shall be no effect on the set, nor will an error be returned.

FD_ZERO(*fdsetp*) shall initialize the descriptor set pointed to by *fdsetp* to the null set. No error is returned if the set is not empty at the time *FD_ZERO*() is invoked.

The behavior of these macros is undefined if the *fd* argument is less than 0 or greater than or equal to FD_SETSIZE, or if *fd* is not a valid file descriptor, or if any of the arguments are expressions with side effects.

Return Value

Upon successful completion, the *pselect*() and *select*() functions shall return the total number of bits set in the bit masks. Otherwise, -1 shall be returned, and *errno* shall be set to indicate the error.

FD_CLR(), *FD_SET*(), and *FD_ZERO*() do not return a value. *FD_ISSET*() shall return a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and 0 otherwise.

Errors

Under the following conditions, *pselect*() and *select*() shall fail and set *errno* to:

EBADF

One or more of the file descriptor sets specified a file descriptor that is not a valid open file descriptor.

EINTR

The function was interrupted before any of the selected events occurred and before the timeout interval expired.

If SA_RESTART has been set for the interrupting signal, it is implementation-defined whether the function restarts or returns with [EINTR].

EINVAL

An invalid timeout interval was specified.

EINVAL

The *nfds* argument is less than 0 or greater than FD_SETSIZE.

EINVAL

One of the specified file descriptors refers to a STREAM or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.

The following sections are informative.

Examples

None.

Application Usage

None.

Rationale

In previous versions of the Single UNIX Specification, the *select*() function was defined in the <*sys/time.h*> header. This is now changed to <*sys/select.h*>. The rationale for this change was as follows: the introduction of the *pselect*() function included the <*sys/select.h*> header and the <*sys/select.h*> header defines all the related definitions for the *pselect*() and *select*() functions. Backwards-compatibility to existing XSI implementations is handled by allowing <*sys/time.h*> to include <*sys/select.h*>.

Future Directions

None.

See Also

accept(), *alarm*(), *connect*(), *fcntl*(), *poll*(), *read*(), *recvmsg*(), *sendmsg*(), *setitimer*(), *ualarm*(), *write*(), the Base Definitions volume of IEEE Std 1003.1-2001, <*sys/select.h*>, <*sys/time.h*>

Copyright

System Interface (POSIX), The Open Group Base Specifications Issue 6, Copyright Â© 2001-2003 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Referenced By

curl_multi_perform(3), **libssh2_poll**(3), **vga_waitevent**(3), **xorg.conf**(5), **zshmodules**(1)