Select()

SYNOPSIS

```
/* According to POSIX.1-2001, POSIX.1-2008 */
#include <sys/select.h>
/* According to earlier standards */
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
int select(int nfds, fd_set *readfds, fd_set *writefds,
                fd_set *exceptfds, struct timeval *timeout);
void FD_CLR(int fd, fd_set *set);
int  FD_ISSET(int fd, fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_ZERO(fd_set *set);

#include <sys/select.h>
int pselect(int nfds, fd_set *readfds, fd_set *writefds,
                fd_set *exceptfds, const struct timespec *timeout,
                const sigset_t *sigmask);
```

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

pselect(): _POSIX_C_SOURCE >= 200112L

DESCRIPTION

select() and pselect() allow a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become "ready" for some class of I/O operation. A file descriptor is considered ready if it is possible to perform a corresponding I/O operation.

select() can monitor only file descriptors numbers that are less than FD_SET-SIZE

The operation of select() and pselect() is identical, other than these three differences:

(i) select() uses a timeout that is a struct timeval (with seconds and microseconds), while pselect() uses a struct timespec (with seconds and nanoseconds).

(ii) select() may update the timeout argument to indicate how much time was left. pselect() does not change this argument.

(iii) select() has no sigmask argument, and behaves as pselect() called with NULL sigmask.

RETURN VALUE

On success, select() and pselect() return the number of file descriptors contained in the three returned descriptor sets (that is, the total number of bits that are set in readfds, writefds, exceptfds) which may be zero if the timeout expires before anything interesting happens. On error, -1 is returned, and errno is set to indicate the error; the file descriptor sets are unmodified, and timeout becomes undefined.

Poll()

SYNOPSIS

```
#include <poll.h>

int poll(struct pollfd *fds, nfds_t nfds, int timeout);


#define _GNU_SOURCE             /* See feature_test_macros(7) */

#include <signal.h>

#include <poll.h>

int ppoll(struct pollfd *fds, nfds_t nfds,

        const struct timespec *tmo_p, const sigset_t *sigmask);
```

DESCRIPTION

poll() performs a similar task to select(2): it waits for one of a set of file descriptors to become ready to perform I/O.

The set of file descriptors to be monitored is specified in the fds argument,

```
struct pollfd {
    int     fd;             /* file descriptor */
    short events;       /* requested events */
    short revents;      /* returned events */
};
```

The caller should specify the number of items in the fds array in nfds.

ppoll()

The relationship between poll() and ppoll() is analogous to the relationship between select(2) and pselect(2): like pselect(2), ppoll() allows an application to safely wait until either a file descriptor becomes ready or until a signal is caught.

RETURN VALUE

On success, a positive number is returned; this is the number of structures which have nonzero revents fields (in other words, those descriptors with events or errors reported). A value of 0 indicates that the call timed out and no file descriptors were ready. On error, -1 is returned, and errno is set appropriately.

Epoll_wait()

SYNOPSIS

#include <sys/epoll.h>

int epoll_wait(int epfd, struct epoll_event *events, int maxevents, int timeout);

int epoll_pwait(int epfd, struct epoll_event *events, int maxevents, int timeout, const sigset_t *sigmask);

The epoll_wait() system call waits for events on the epoll(7) instance referred to by the file descriptor epfd. The memory area pointed to by events will contain the events that will be available for the caller. Up to max-events are returned by epoll_wait(). The maxevents argument must be greater than zero.

epoll_pwait()

The relationship between epoll_wait() and epoll_pwait() is analogous to the relationship between select(2) and pselect(2): like pselect(2), epoll_pwait() allows an application to safely wait until either a file descriptor becomes ready or until a signal is caught.

RETURN VALUE

When successful, epoll_wait() returns the number of file descriptors ready for the requested I/O, or zero if no file descriptor became ready during the requested timeout milliseconds. When an error occurs, epoll_wait() returns -1 and errno is set appropriately.