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 'poll.3p' command

$ man poll.3p

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

poll ? input/output multiplexing

SYNOPSIS

#include <poll.h>

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

DESCRIPTION

The poll() function provides applications with a mechanism for multi? plexing input/output over a set of file descriptors. For each member of the array pointed to by fds, poll() shall examine the given file de? scriptor for the event(s) specified in events. The number of pollfd structures in the fds array is specified by nfds. The poll() function shall identify those file descriptors on which an application can read or write data, or on which certain events have occurred.

The fds argument specifies the file descriptors to be examined and the events of interest for each file descriptor. It is a pointer to an ar? ray with one member for each open file descriptor of interest. The ar? ray's members are pollfd structures within which fd specifies an open

file descriptor and events and revents are bitmasks constructed by OR'ing a combination of the following event flags:

POLLIN Data other than high-priority data may be read without blocking.

For STREAMS, this flag is set in revents even if the mes? sage is of zero length. This flag shall be equivalent to POLLRDNORM | POLLRDBAND.

POLLRDNORM Normal data may be read without blocking.

For STREAMS, data on priority band 0 may be read without blocking. This flag is set in revents even if the message is of zero length.

POLLRDBAND Priority data may be read without blocking.

For STREAMS, data on priority bands greater than 0 may be read without blocking. This flag is set in revents even if the message is of zero length.

POLLPRI High-priority data may be read without blocking.

For STREAMS, this flag is set in revents even if the mes? sage is of zero length.

POLLOUT Normal data may be written without blocking.

For STREAMS, data on priority band 0 may be written without blocking.

POLLWRNORM Equivalent to POLLOUT.

POLLWRBAND Priority data may be written.

For STREAMS, data on priority bands greater than 0 may be written without blocking. If any priority band has been written to on this STREAM, this event only examines bands that have been written to at least once.

POLLERR An error has occurred on the device or stream. This flag is only valid in the revents bitmask; it shall be ignored in the events member.

POLLHUP A device has been disconnected, or a pipe or FIFO has been closed by the last process that had it open for writing.

Once set, the hangup state of a FIFO shall persist until

some process opens the FIFO for writing or until all read-
only file descriptors for the FIFO are closed. This event
and POLLOUT are mutually-exclusive; a stream can never be
writable if a hangup has occurred. However, this event and
POLLIN, POLLRDNORM, POLLRDBAND, or POLLPRI are not mutu?
ally-exclusive. This flag is only valid in the revents bit?
mask; it shall be ignored in the events member.

POLLNVAL The specified fd value is invalid. This flag is only valid
in the revents member; it shall ignored in the events mem?
ber.

The significance and semantics of normal, priority, and high-priority data are file and device-specific.

If the value of fd is less than 0, events shall be ignored, and revents shall be set to 0 in that entry on return from poll().

In each pollfd structure, poll() shall clear the revents member, except that where the application requested a report on a condition by setting one of the bits of events listed above, poll() shall set the corre? sponding bit in revents if the requested condition is true. In addi? tion, poll() shall set the POLLHUP, POLLERR, and POLLNVAL flag in revents if the condition is true, even if the application did not set the corresponding bit in events.

If none of the defined events have occurred on any selected file de? scriptor, poll() shall wait at least timeout milliseconds for an event to occur on any of the selected file descriptors. If the value of time? out is 0, poll() shall return immediately. If the value of timeout is -1, poll() shall block until a requested event occurs or until the call is interrupted.

Implementations may place limitations on the granularity of timeout in? tervals. 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.

The poll() function shall not be affected by the O_NONBLOCK flag.

The poll() function shall support regular files, terminal and pseudo-

terminal devices, FIFOs, pipes, sockets and STREAMS-based files. The behavior of poll() on elements of fds that refer to other types of file is unspecified.

Regular files shall always poll TRUE for reading and writing.

A file descriptor for a socket that is listening for connections shall indicate that it is ready for reading, once connections are available.

A file descriptor for a socket that is connecting asynchronously shall indicate that it is ready for writing, once a connection has been es? tablished.

Provided the application does not perform any action that results in unspecified or undefined behavior, the value of the fd and events mem? bers of each element of fds shall not be modified by poll().

RETURN VALUE

Upon successful completion, poll() shall return a non-negative value. A positive value indicates the total number of pollfd structures that have selected events (that is, those for which the revents member is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, poll() shall return -1 and set errno to indicate the error.

ERRORS

The poll() function shall fail if:

EAGAIN The allocation of internal data structures failed but a subse? quent request may succeed.

EINTR A signal was caught during poll().

EINVAL The nfds argument is greater than {OPEN_MAX}, or one of the fd members refers to a STREAM or multiplexer that is linked (di? rectly or indirectly) downstream from a multiplexer.

The following sections are informative.

EXAMPLES

Checking for Events on a Stream

The following example opens a pair of STREAMS devices and then waits for either one to become writable. This example proceeds as follows:

1. Sets the timeout parameter to 500 milliseconds.

2. Opens the STREAMS devices /dev/dev0 and /dev/dev1, and then polls them, specifying POLLOUT and POLLWRBAND as the events of interest. The STREAMS device names /dev/dev0 and /dev/dev1 are only examples of how STREAMS devices can be named; STREAMS naming conventions may vary among systems conforming to the POSIX.1?2008.

3. Uses the ret variable to determine whether an event has occurred on either of the two STREAMS. The poll() function is given 500 mil? liseconds to wait for an event to occur (if it has not occurred prior to the poll() call).

4. Checks the returned value of ret. If a positive value is returned, one of the following can be done:

   a. Priority data can be written to the open STREAM on priority bands greater than 0, because the POLLWRBAND event occurred on the open STREAM (fds[0] or fds[1]).

   b. Data can be written to the open STREAM on priority-band 0, be? cause the POLLOUT event occurred on the open STREAM (fds[0] or fds[1]).

5. If the returned value is not a positive value, permission to write data to the open STREAM (on any priority band) is denied.

6. If the POLLHUP event occurs on the open STREAM (fds[0] or fds[1]), the device on the open STREAM has disconnected.

```
#include <stropts.h>
#include <poll.h>
...
struct pollfd fds[2];
int timeout_msecs = 500;
int ret;
    int i;
/* Open STREAMS device. */
fds[0].fd = open("/dev/dev0", ...);
fds[1].fd = open("/dev/dev1", ...);
fds[0].events = POLLOUT | POLLWRBAND;
fds[1].events = POLLOUT | POLLWRBAND;
```

```
ret = poll(fds, 2, timeout_msecs);
if (ret > 0) {
    /* An event on one of the fds has occurred. */
    for (i=0; i<2; i++) {
        if (fds[i].revents & POLLWRBAND) {
        /* Priority data may be written on device number i. */
...
        }
        if (fds[i].revents & POLLOUT) {
        /* Data may be written on device number i. */
...
        }
        if (fds[i].revents & POLLHUP) {
        /* A hangup has occurred on device number i. */
...
        }
    }
}
```

APPLICATION USAGE

None.

RATIONALE

The POLLHUP event does not occur for FIFOs just because the FIFO is not open for writing. It only occurs when the FIFO is closed by the last writer and persists until some process opens the FIFO for writing or until all read-only file descriptors for the FIFO are closed.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6, STREAMS, getmsg(), pselect(), putmsg(), read(), write()

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

table Operating System Interface (POSIX), The Open Group Base Specifi? cations Issue 7, 2018 Edition, Copyright (C) 2018 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 .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker? nel.org/doc/man-pages/reporting_bugs.html .