



Rocky Enterprise Linux 9.2 Manual Pages on command 'ppoll.2'

C:\>man ppoll.2

POLL(2) Linux Programmer's Manual POLL(2)

NAME

poll, ppoll - wait for some event on a file descriptor

SYNOPSIS

```
#include <poll.h>

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

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

#include <signal.h>

#include <poll.h>

int ppoll(struct pollfd *fds, nfds_t nfd,
          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, which is an array of structures of the following form:

```
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 nfd.

The field `fd` contains a file descriptor for an open file. If this field is negative, then the corresponding events field is ignored and the `revents` field returns zero. (This provides an easy way of ignoring a file descriptor for a single `poll()` call: simply negate the `fd` field. Note, however, that this technique can't be used to ignore file descriptor 0.)

The field `events` is an input parameter, a bit mask specifying the events the application is interested in for the file descriptor `fd`. This field may be specified as zero, in which case the only events that can be returned in `revents` are `POLLHUP`, `POLLERR`, and `POLLNVAL` (see below).

The field `revents` is an output parameter, filled by the kernel with the events that actually occurred. The bits returned in `revents` can include any of those specified in `events`, or one of the values `POLLERR`, `POLLHUP`, or `POLLNVAL`. (These three bits are meaningless in the `events` field, and will be set in the `revents` field whenever the corresponding condition is true.)

If none of the events requested (and no error) has occurred for any of the file descriptors, then `poll()` blocks until one of the events occurs.

The `timeout` argument specifies the number of milliseconds that `poll()` should block waiting for a file descriptor to become ready. The call will block until either:

- * a file descriptor becomes ready;
- * the call is interrupted by a signal handler; or
- * the timeout expires.

Note that the timeout interval will be rounded up to the system clock granularity, and kernel scheduling delays mean that the blocking interval may be overrun by a small amount. Specifying a negative value in `timeout` means an infinite timeout. Specifying a `timeout` of zero causes `poll()` to return immediately, even if no file descriptors are ready.

The bits that may be set/returned in `events` and `revents` are defined in `<poll.h>`:

`POLLIN` There is data to read.

`POLLPRI`

There is some exceptional condition on the file descriptor. Possibilities include:

- * There is out-of-band data on a TCP socket (see `tcp(7)`).
- * A pseudoterminal master in packet mode has seen a state change on the

slave (see `ioctl_tty(2)`).

* A `cgroup.events` file has been modified (see `cgroups(7)`).

POLLOUT

Writing is now possible, though a write larger than the available space in a socket or pipe will still block (unless `O_NONBLOCK` is set).

POLLRDHUP (since Linux 2.6.17)

Stream socket peer closed connection, or shut down writing half of connection. The `_GNU_SOURCE` feature test macro must be defined (before including any header files) in order to obtain this definition.

POLLERR

Error condition (only returned in revents; ignored in events). This bit is also set for a file descriptor referring to the write end of a pipe when the read end has been closed.

POLLHUP

Hang up (only returned in revents; ignored in events). Note that when reading from a channel such as a pipe or a stream socket, this event merely indicates that the peer closed its end of the channel. Subsequent reads from the channel will return 0 (end of file) only after all outstanding data in the channel has been consumed.

POLLNVAL

Invalid request: fd not open (only returned in revents; ignored in events).

When compiling with `_XOPEN_SOURCE` defined, one also has the following, which convey no further information beyond the bits listed above:

POLLRDNORM

Equivalent to `POLLIN`.

POLLRDBAND

Priority band data can be read (generally unused on Linux).

POLLWRNORM

Equivalent to `POLLOUT`.

POLLWRBAND

Priority data may be written.

Linux also knows about, but does not use `POLLMSG`.

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.

Other than the difference in the precision of the timeout argument, the following `ppoll()` call:

```
ready = ppoll(&fds, nfd, tmo_p, &sigmask);
```

is nearly equivalent to atomically executing the following calls:

```
sigset_t origmask;
```

```
int timeout;
```

```
timeout = (tmo_p == NULL) ? -1 :
```

```
(tmo_p->tv_sec * 1000 + tmo_p->tv_nsec / 1000000);
```

```
pthread_sigmask(SIG_SETMASK, &sigmask, &origmask);
```

```
ready = poll(&fds, nfd, timeout);
```

```
pthread_sigmask(SIG_SETMASK, &origmask, NULL);
```

The above code segment is described as nearly equivalent because whereas a negative timeout value for `poll()` is interpreted as an infinite timeout, a negative value expressed in `*tmo_p` results in an error from `ppoll()`.

See the description of `pselect(2)` for an explanation of why `ppoll()` is necessary.

If the `sigmask` argument is specified as `NULL`, then no signal mask manipulation is performed (and thus `ppoll()` differs from `poll()` only in the precision of the timeout argument).

The `tmo_p` argument specifies an upper limit on the amount of time that `ppoll()` will block. This argument is a pointer to a structure of the following form:

```
struct timespec {  
    long   tv_sec;    /* seconds */  
    long   tv_nsec;   /* nanoseconds */  
};
```

If `tmo_p` is specified as `NULL`, then `ppoll()` can block indefinitely.

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 de-

scriptors were ready. On error, -1 is returned, and `errno` is set appropriately.

ERRORS

`EFAULT` The array given as argument was not contained in the calling program's address space.

`EINTR` A signal occurred before any requested event; see `signal(7)`.

`EINVAL` The `nfds` value exceeds the `RLIMIT_NOFILE` value.

`EINVAL` (`ppoll()`) The timeout value expressed in `*ip` is invalid (negative).

`ENOMEM` There was no space to allocate file descriptor tables.

VERSIONS

The `poll()` system call was introduced in Linux 2.1.23. On older kernels that lack this system call, the glibc (and the old Linux libc) `poll()` wrapper function provides emulation using `select(2)`.

The `ppoll()` system call was added to Linux in kernel 2.6.16. The `ppoll()` library call was added in glibc 2.4.

CONFORMING TO

`poll()` conforms to POSIX.1-2001 and POSIX.1-2008. `ppoll()` is Linux-specific.

NOTES

The operation of `poll()` and `ppoll()` is not affected by the `O_NONBLOCK` flag.

On some other UNIX systems, `poll()` can fail with the error `EAGAIN` if the system fails to allocate kernel-internal resources, rather than `ENOMEM` as Linux does.

POSIX permits this behavior. Portable programs may wish to check for `EAGAIN` and loop, just as with `EINTR`.

Some implementations define the nonstandard constant `INFTIM` with the value -1 for use as a timeout for `poll()`. This constant is not provided in glibc.

For a discussion of what may happen if a file descriptor being monitored by `poll()` is closed in another thread, see `select(2)`.

C library/kernel differences

The Linux `ppoll()` system call modifies its `tmo_p` argument. However, the glibc wrapper function hides this behavior by using a local variable for the timeout argument that is passed to the system call. Thus, the glibc `ppoll()` function does not modify its `tmo_p` argument.

The raw `ppoll()` system call has a fifth argument, `size_t sigsetsize`, which specifies the size in bytes of the `sigmask` argument. The glibc `ppoll()` wrapper function

specifies this argument as a fixed value (equal to `sizeof(kernel_sigset_t)`). See `sigprocmask(2)` for a discussion on the differences between the kernel and the libc notion of the sigset.

BUGS

See the discussion of spurious readiness notifications under the `BUGS` section of `select(2)`.

SEE ALSO

`restart_syscall(2)`, `select(2)`, `select_tut(2)`, `epoll(7)`, `time(7)`

COLOPHON

This page is part of release 5.05 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.

Linux

2019-08-02

POLL(2)