



## ***Linux Ubuntu 26.04 Manual Pages on command 'readv.2'***

### ***\$ man readv.2***

readv(2)                      System Calls Manual                      readv(2)

#### NAME

readv, writev, preadv, pwritev, preadv2, pwritev2 - read or write data into multiple buffers

#### LIBRARY

Standard C library (libc, -lc)

#### SYNOPSIS

```
#include <sys/uio.h>

ssize_t readv(int fd, const struct iovec *iov, int iovcnt);
ssize_t writev(int fd, const struct iovec *iov, int iovcnt);
ssize_t preadv(int fd, const struct iovec *iov, int iovcnt,
               off_t offset);
ssize_t pwritev(int fd, const struct iovec *iov, int iovcnt,
                off_t offset);
ssize_t preadv2(int fd, const struct iovec *iov, int iovcnt,
                off_t offset, int flags);
ssize_t pwritev2(int fd, const struct iovec *iov, int iovcnt,
                 off_t offset, int flags);
```

Feature Test Macro Requirements for glibc (see feature\_test\_macros(7)):

preadv(), pwritev():

Since glibc 2.19:

    \_DEFAULT\_SOURCE

glibc 2.19 and earlier:

    \_BSD\_SOURCE

## DESCRIPTION

The `readv()` system call reads `iovcnt` buffers from the file associated with the file descriptor `fd` into the buffers described by `iov` ("scatter input").

The `writev()` system call writes `iovcnt` buffers of data described by `iov` to the file associated with the file descriptor `fd` ("gather output").

The pointer `iov` points to an array of `iovec` structures, described in `iovec(3type)`.

The `readv()` system call works just like `read(2)` except that multiple buffers are filled.

The `writev()` system call works just like `write(2)` except that multiple buffers are written out.

Buffers are processed in array order. This means that `readv()` completely fills `iov[0]` before proceeding to `iov[1]`, and so on. (If there is insufficient data, then not all buffers pointed to by `iov` may be filled.) Similarly, `writev()` writes out the entire contents of `iov[0]` before proceeding to `iov[1]`, and so on.

The data transfers performed by `readv()` and `writev()` are atomic: the data written by `writev()` is written as a single block that is not intermingled with output from writes in other processes; analogously, `readv()` is guaranteed to read a contiguous block of data from the file, regardless of read operations performed in other threads or processes that have file descriptors referring to the same open file description (see `open(2)`).

### `preadv()` and `pwritev()`

The `preadv()` system call combines the functionality of `readv()` and `pread(2)`. It performs the same task as `readv()`, but adds a fourth argument, `offset`, which specifies the file offset at which the input operation is to be performed.

The `pwritev()` system call combines the functionality of `writev()` and `pwrite(2)`. It performs the same task as `writev()`, but adds a fourth argument, `offset`, which specifies the file offset at which the output operation is to be performed.

The file offset is not changed by these system calls. The file referred to by `fd` must be capable of seeking.

### `preadv2()` and `pwritev2()`

These system calls are similar to `preadv()` and `pwritev()` calls, but add a fifth argument, `flags`, which modifies the behavior on a per-call basis.

Unlike `preadv()` and `pwritev()`, if the `offset` argument is `-1`, then the current file offset is used and updated.

The `flags` argument contains a bitwise OR of zero or more of the following flags:

#### RWF\_DSYNC (since Linux 4.7)

Provide a per-write equivalent of the O\_DSYNC open(2) flag. This flag is meaningful only for pwritev2(), and its effect applies only to the data range written by the system call.

#### RWF\_HIPRI (since Linux 4.6)

High priority read/write. Allows block-based filesystems to use polling of the device, which provides lower latency, but may use additional resources. (Currently, this feature is usable only on a file descriptor opened using the O\_DIRECT flag.)

#### RWF\_SYNC (since Linux 4.7)

Provide a per-write equivalent of the O\_SYNC open(2) flag. This flag is meaningful only for pwritev2(), and its effect applies only to the data range written by the system call.

#### RWF\_NOWAIT (since Linux 4.14)

Do not wait for data which is not immediately available. If this flag is specified, the preadv2() system call will return instantly if it would have to read data from the backing storage or wait for a lock. If some data was successfully read, it will return the number of bytes read. If no bytes were read, it will return -1 and set errno to EAGAIN (but see BUGS). Currently, this flag is meaningful only for preadv2().

#### RWF\_APPEND (since Linux 4.16)

Provide a per-write equivalent of the O\_APPEND open(2) flag. This flag is meaningful only for pwritev2(), and its effect applies only to the data range written by the system call. The offset argument does not affect the write operation; the data is always appended to the end of the file. However, if the offset argument is -1, the current file offset is updated.

#### RWF\_NOAPPEND (since Linux 6.9)

Do not honor the O\_APPEND open(2) flag. This flag is meaningful only for pwritev2(). Historically, Linux honored O\_APPEND flag if set and ignored the offset argument, which is a bug. For pwritev2(), the offset argument is honored as expected if RWF\_NOAPPEND flag is set, the same as if O\_APPEND flag were not set.

#### RWF\_ATOMIC (since Linux 6.11)

Requires that writes to regular files in block-based filesystems be issued with torn-write protection. Torn-write protection means that for a power or any other hardware

failure, all or none of the data from the write will be stored, but never a mix of old and new data. This flag is meaningful only for `pwritev2()`, and its effect applies only to the data range written by the system call. The total write length must be power-of-2 and must be sized in the range `[stx_atomic_write_unit_min, stx_atomic_write_unit_max]`. The write must be at a naturally-aligned offset within the file with respect to the total write length. For example, a write of length 32KiB at a file offset of 32KiB is permitted, however a write of length 32KiB at a file offset of 48KiB is not permitted. The upper limit of `iovcnt` for `pwritev2()` is given by the value in `stx_atomic_write_segments_max`. Torn-write protection only works with `O_DIRECT` flag, that is, buffered writes are not supported. To guarantee consistency from the write between a file's in-core state with the storage device, `O_SYNC` or `O_DSYNC` must be specified for `open(2)`. The same synchronized I/O guarantees as described in `open(2)` are provided when these flags or their equivalent flags and system calls are used (e.g., if `RWF_SYNC` is specified for `pwritev2()`).

#### `RWF_DONTCACHE` (since Linux 6.14)

Reads or writes to a regular file will prune instantiated page cache content when the operation completes. This is different than normal buffered I/O, where the data usually remains in cache until such time that it gets reclaimed due to memory pressure. If ranges of the read or written I/O were already in cache before this read or write, then those ranges will not be pruned at I/O completion time.

Additionally, any range dirtied by a write operation with `RWF_DONTCACHE` set will get kicked off for writeback. This is similar to calling `sync_file_range(2)` with `SYNC_FILE_RANGE_WRITE` to start writeback on the given range. `RWF_DONTCACHE` is a hint, or best effort, where no hard guarantees are given on the state of the page cache once the operation completes.

If used on a file system or block device that doesn't support it, it will return -1, and `errno` will be set to `EOPNOTSUPP`.

#### RETURN VALUE

On success, `readv()`, `preadv()`, and `preadv2()` return the number of bytes read; `writev()`, `pwritev()`, and `pwritev2()` return the number of bytes written.

Note that it is not an error for a successful call to transfer fewer bytes than requested (see `read(2)` and `write(2)`).

On error, -1 is returned, and `errno` is set to indicate the error.

## ERRORS

The errors are as given for `read(2)` and `write(2)`. Furthermore, `preadv()`, `preadv2()`, `pwritev()`, and `pwritev2()` can also fail for the same reasons as `lseek(2)`. Additionally, the following errors are defined:

**EINVAL** The sum of the `iov_len` values overflows an `ssize_t` value.

**EINVAL** If `RWF_ATOMIC` is specified, the combination of the sum of the `iov_len` values and the offset value does not comply with the length and offset torn-write protection rules.

**EINVAL** The vector count, `iovcnt`, is less than zero or greater than the permitted maximum.

If `RWF_ATOMIC` is specified, this maximum is given by the `stx_atomic_write_segments_max` value from `statx`.

**EOPNOTSUPP**

`RWF_DONTCACHE` was set in flags and the file doesn't support it.

**EOPNOTSUPP**

An unknown flag is specified in flags.

## VERSIONS

### C library/kernel differences

The raw `preadv()` and `pwritev()` system calls have call signatures that differ slightly from that of the corresponding GNU C library wrapper functions shown in the SYNOPSIS. The final argument, `offset`, is unpacked by the wrapper functions into two arguments in the system calls:

`unsigned long pos_l, unsigned long pos`

These arguments contain, respectively, the low order and high order 32 bits of `offset`.

## STANDARDS

`readv()`

`writev()`

POSIX.1-2024.

`preadv()`

`pwritev()`

BSD.

`preadv2()`

`pwritev2()`

Linux.

## HISTORY

readv()

writev()

POSIX.1-2001, 4.4BSD (first appeared in 4.2BSD).

preadv(), pwritev(): Linux 2.6.30, glibc 2.10.

preadv2(), pwritev2(): Linux 4.6, glibc 2.26.

#### Historical C library/kernel differences

To deal with the fact that IOV\_MAX was so low on early versions of Linux, the glibc wrapper functions for readv() and writev() did some extra work if they detected that the underlying kernel system call failed because this limit was exceeded. In the case of readv(), the wrapper function allocated a temporary buffer large enough for all of the items specified by iov, passed that buffer in a call to read(2), copied data from the buffer to the locations specified by the iov\_base fields of the elements of iov, and then freed the buffer. The wrapper function for writev() performed the analogous task using a temporary buffer and a call to write(2).

The need for this extra effort in the glibc wrapper functions went away with Linux 2.2 and later. However, glibc continued to provide this behavior until glibc 2.10. Starting with glibc 2.9, the wrapper functions provide this behavior only if the library detects that the system is running a Linux kernel older than Linux 2.6.18 (an arbitrarily selected kernel version). And since glibc 2.20 (which requires a minimum of Linux 2.6.32), the glibc wrapper functions always just directly invoke the system calls.

#### NOTES

POSIX.1 allows an implementation to place a limit on the number of items that can be passed in iov. An implementation can advertise its limit by defining IOV\_MAX in <limits.h> or at run time via the return value from sysconf(\_SC\_IOV\_MAX). On modern Linux systems, the limit is 1024. Back in Linux 2.0 days, this limit was 16.

#### BUGS

Linux 5.9 and Linux 5.10 have a bug where preadv2() with the RWF\_NOWAIT flag may return 0 even when not at end of file.

#### EXAMPLES

The following code sample demonstrates the use of writev():

```
char    *str0 = "hello ";
char    *str1 = "world\n";
ssize_t  nwritten;
```

```
struct iovec iov[2];  
iov[0].iov_base = str0;  
iov[0].iov_len = strlen(str0);  
iov[1].iov_base = str1;  
iov[1].iov_len = strlen(str1);  
nwritten = writev(STDOUT_FILENO, iov, 2);
```

#### SEE ALSO

[pread\(2\)](#), [read\(2\)](#), [write\(2\)](#)

Linux man-pages 6.17

2026-02-08

[readv\(2\)](#)