



Rocky Enterprise Linux 9.2 Manual Pages on command 'dup2.2'

\$ man dup2.2

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

NAME

dup, dup2, dup3 - duplicate a file descriptor

SYNOPSIS

```
#include <unistd.h>

int dup(int oldfd);

int dup2(int oldfd, int newfd);

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

#include <fcntl.h>           /* Obtain O_* constant definitions */

#include <unistd.h>

int dup3(int oldfd, int newfd, int flags);
```

DESCRIPTION

The `dup()` system call creates a copy of the file descriptor `oldfd`, using the lowest-numbered unused file descriptor for the new descriptor. After a successful return, the old and new file descriptors may be used interchangeably. They refer to the same open file description (see `open(2)`) and thus share file offset and file status flags; for example, if the file offset is modified by using `lseek(2)` on one of the file de?

scriptors, the offset is also changed for the other.

The two file descriptors do not share file descriptor flags (the close-on-exec flag). The close-on-exec flag (FD_CLOEXEC; see `fcntl(2)`) for the duplicate descriptor is off.

dup2()

The `dup2()` system call performs the same task as `dup()`, but instead of using the lowest-numbered unused file descriptor, it uses the file descriptor number specified in `newfd`. If the file descriptor `newfd` was previously open, it is silently closed before being reused.

The steps of closing and reusing the file descriptor `newfd` are performed atomically. This is important, because trying to implement equivalent functionality using `close(2)` and `dup()` would be subject to race conditions, whereby `newfd` might be reused between the two steps. Such reuse could happen because the main program is interrupted by a signal handler that allocates a file descriptor, or because a parallel thread allocates a file descriptor.

Note the following points:

- * If `oldfd` is not a valid file descriptor, then the call fails, and `newfd` is not closed.
- * If `oldfd` is a valid file descriptor, and `newfd` has the same value as `oldfd`, then `dup2()` does nothing, and returns `newfd`.

dup3()

`dup3()` is the same as `dup2()`, except that:

- * The caller can force the close-on-exec flag to be set for the new file descriptor by specifying `O_CLOEXEC` in `flags`. See the description of the same flag in `open(2)` for reasons why this may be useful.
- * If `oldfd` equals `newfd`, then `dup3()` fails with the error `EINVAL`.

RETURN VALUE

On success, these system calls return the new file descriptor. On error, -1 is returned, and `errno` is set appropriately.

ERRORS

`EBADF` `oldfd` isn't an open file descriptor.

`EBADF` `newfd` is out of the allowed range for file descriptors (see the

discussion of RLIMIT_NOFILE in getrlimit(2)).

EBUSY (Linux only) This may be returned by dup2() or dup3() during a race condition with open(2) and dup().

EINTR The dup2() or dup3() call was interrupted by a signal; see signal(7).

EINVAL (dup3()) flags contain an invalid value.

EINVAL (dup3()) oldfd was equal to newfd.

EMFILE The per-process limit on the number of open file descriptors has been reached (see the discussion of RLIMIT_NOFILE in getrlimit(2)).

VERSIONS

dup3() was added to Linux in version 2.6.27; glibc support is available starting with version 2.9.

CONFORMING TO

dup(), dup2(): POSIX.1-2001, POSIX.1-2008, SVr4, 4.3BSD.

dup3() is Linux-specific.

NOTES

The error returned by dup2() is different from that returned by fcntl(..., F_DUPFD, ...)

when newfd is out of range. On some systems,

dup2() also sometimes returns EINVAL like F_DUPFD.

If newfd was open, any errors that would have been reported at close(2)

time are lost. If this is of concern, then unless the program is single-threaded and does not allocate file descriptors in signal handlers?

the correct approach is not to close newfd before calling dup2(), because of the race condition described above. Instead, code something

like the following could be used:

```
/* Obtain a duplicate of 'newfd' that can subsequently
```

```
be used to check for close() errors; an EBADF error
```

```
means that 'newfd' was not open. */
```

```
tmpfd = dup(newfd);
```

```
if (tmpfd == -1 && errno != EBADF) {
```

```
    /* Handle unexpected dup() error */
```

```
}
```

```

/* Atomically duplicate 'oldfd' on 'newfd' */
if (dup2(oldfd, newfd) == -1) {
    /* Handle dup2() error */
}

/* Now check for close() errors on the file originally
   referred to by 'newfd' */
if (tmpfd != -1) {
    if (close(tmpfd) == -1) {
        /* Handle errors from close */
    }
}
}

```

SEE ALSO

close(2), fcntl(2), open(2), pidfd_getfd(2)

COLOPHON

This page is part of release 5.10 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/>.