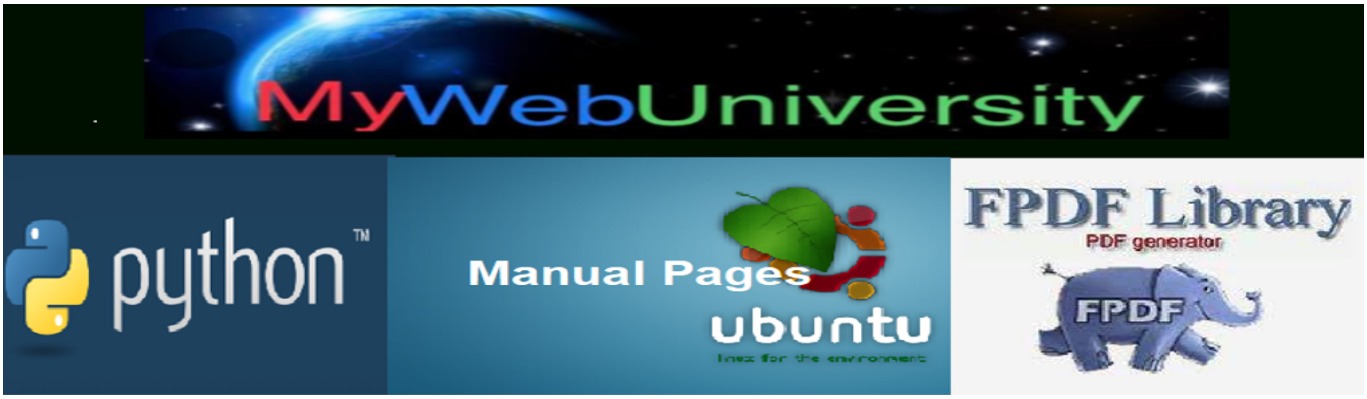

Rocky Enterprise Linux 9.2 Manual Pages on command 'copy_file_range.2'

$ man copy_file_range.2

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

NAME

copy_file_range - Copy a range of data from one file to another

SYNOPSIS

```
#define _GNU_SOURCE
#include <unistd.h>
ssize_t copy_file_range(int fd_in, loff_t *off_in,
                        int fd_out, loff_t *off_out,
                        size_t len, unsigned int flags);
```

DESCRIPTION

The copy_file_range() system call performs an in-kernel copy between two file descriptors without the additional cost of transferring data from the kernel to user space and then back into the kernel. It copies up to len bytes of data from the source file descriptor fd_in to the target file descriptor fd_out, overwriting any data that exists within the requested range of the target file.

The following semantics apply for off_in, and similar statements apply to off_out:

* If off_in is NULL, then bytes are read from fd_in starting from the file offset, and the file offset is adjusted by the number of bytes copied.
* If off_in is not NULL, then off_in must point to a buffer that specifies the starting offset where bytes from fd_in will be read. The file offset of fd_in is not changed, but off_in is adjusted appropriately.

fd_in and fd_out can refer to the same file. If they refer to the same file, then the source and target ranges are not allowed to overlap.

The flags argument is provided to allow for future extensions and currently must be set to 0.

RETURN VALUE

Upon successful completion, copy_file_range() will return the number of bytes copied between files. This could be less than the length originally requested. If the file offset of fd_in is at or past the end of file, no bytes are copied, and copy_file_range() returns zero.

On error, copy_file_range() returns -1 and errno is set to indicate the error.

ERRORS

EBADF One or more file descriptors are not valid.

EBADF fd_in is not open for reading; or fd_out is not open for writing.

EBADF The O_APPEND flag is set for the open file description (see open(2)) referred to by the file descriptor fd_out.

EFBIG An attempt was made to write at a position past the maximum file offset the kernel supports.

EFBIG An attempt was made to write a range that exceeds the allowed maximum file size. The maximum file size differs between filesystem implementations and can be different from the maximum allowed file offset.

EFBIG An attempt was made to write beyond the process's file size resource limit. This may also result in the process receiving a SIGXFSZ signal.

EINVAL The flags argument is not 0.

EINVAL fd_in and fd_out refer to the same file and the source and target ranges overlap.

EINVAL Either fd_in or fd_out is not a regular file.

EIO A low-level I/O error occurred while copying.

EISDIR Either fd_in or fd_out refers to a directory.

ENOMEM Out of memory.

ENOSPC There is not enough space on the target filesystem to complete the copy.

EOVERFLOW

The requested source or destination range is too large to represent in the specified data types.

EPERM fd_out refers to an immutable file.

ETXTBSY

Either fd_in or fd_out refers to an active swap file.

EXDEV The files referred to by fd_in and fd_out are not on the same mounted filesystem (pre Linux 5.3).

VERSIONS

The copy_file_range() system call first appeared in Linux 4.5, but glibc 2.27 provides a user-space emulation when it is not available.

A major rework of the kernel implementation occurred in 5.3. Areas of the API that weren't clearly defined were clarified and the API bounds are much more strictly checked than on earlier kernels. Applications should target the behaviour and requirements of 5.3 kernels.

First support for cross-filesystem copies was introduced in Linux 5.3. Older kernels will return -EXDEV when cross-filesystem copies are attempted.

CONFORMING TO

The copy_file_range() system call is a nonstandard Linux and GNU extension.

NOTES

If fd_in is a sparse file, then copy_file_range() may expand any holes existing in the re? quested range. Users may benefit from calling copy_file_range() in a loop, and using the lseek(2) SEEK_DATA and SEEK_HOLE operations to find the locations of data segments.

copy_file_range() gives filesystems an opportunity to implement "copy acceleration" tech? niques, such as the use of reflinks (i.e., two or more inodes that share pointers to the same copy-on-write disk blocks) or server-side-copy (in the case of NFS).

EXAMPLES

```
#define _GNU_SOURCE
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/syscall.h>
#include <unistd.h>
/* On versions of glibc before 2.27, we must invoke copy_file_range()
   using syscall(2) */
static loff_t
copy_file_range(int fd_in, loff_t *off_in, int fd_out,
                loff_t *off_out, size_t len, unsigned int flags)
```

```
{
    return syscall(__NR_copy_file_range, fd_in, off_in, fd_out,
                   off_out, len, flags);
}
int
main(int argc, char **argv)
{
    int fd_in, fd_out;
    struct stat stat;
    loff_t len, ret;
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <source> <destination>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    fd_in = open(argv[1], O_RDONLY);
    if (fd_in == -1) {
        perror("open (argv[1])");
        exit(EXIT_FAILURE);
    }
    if (fstat(fd_in, &stat) == -1) {
        perror("fstat");
        exit(EXIT_FAILURE);
    }
    len = stat.st_size;
    fd_out = open(argv[2], O_CREAT | O_WRONLY | O_TRUNC, 0644);
    if (fd_out == -1) {
        perror("open (argv[2])");
        exit(EXIT_FAILURE);
    }
    do {
        ret = copy_file_range(fd_in, NULL, fd_out, NULL, len, 0);
        if (ret == -1) {
            perror("copy_file_range");
```

```
            exit(EXIT_FAILURE);
        }
        len -= ret;
    } while (len > 0 && ret > 0);
    close(fd_in);
    close(fd_out);
    exit(EXIT_SUCCESS);
}
```

SEE ALSO

lseek(2), sendfile(2), splice(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/.