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 'fflush.3p' command

$ man fflush.3p

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

fflush ? flush a stream

SYNOPSIS

#include <stdio.h>

int fflush(FILE *stream);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1?2017 defers to the ISO C standard.

If stream points to an output stream or an update stream in which the most recent operation was not input, fflush() shall cause any unwritten data for that stream to be written to the file, and the last data modi? fication and last file status change timestamps of the underlying file shall be marked for update.

For a stream open for reading with an underlying file description, if the file is not already at EOF, and the file is one capable of seeking,

the file offset of the underlying open file description shall be set to the file position of the stream, and any characters pushed back onto the stream by ungetc() or ungetwc() that have not subsequently been read from the stream shall be discarded (without further changing the file offset).

If stream is a null pointer, fflush() shall perform this flushing ac? tion on all streams for which the behavior is defined above.

RETURN VALUE

Upon successful completion, fflush() shall return 0; otherwise, it shall set the error indicator for the stream, return EOF, and set errno to indicate the error.

ERRORS

The fflush() function shall fail if:

EAGAIN The O_NONBLOCK flag is set for the file descriptor underlying stream and the thread would be delayed in the write operation.

EBADF The file descriptor underlying stream is not valid.

EFBIG An attempt was made to write a file that exceeds the maximum file size.

EFBIG An attempt was made to write a file that exceeds the file size limit of the process.

EFBIG The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

EINTR The fflush() function was interrupted by a signal.

EIO The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined condi? tions.

ENOMEM The underlying stream was created by open_memstream() or open_wmemstream() and insufficient memory is available.

ENOSPC There was no free space remaining on the device containing the

file or in the buffer used by the fmemopen() function.

EPIPE An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal shall also be sent to the thread.

The fflush() function may fail if:

ENXIO A request was made of a nonexistent device, or the request was outside the capabilities of the device.

The following sections are informative.

EXAMPLES

Sending Prompts to Standard Output

The following example uses printf() calls to print a series of prompts for information the user must enter from standard input. The fflush() calls force the output to standard output. The fflush() function is used because standard output is usually buffered and the prompt may not immediately be printed on the output or terminal. The getline() func? tion calls read strings from standard input and place the results in variables, for use later in the program.

```
char *user;
char *oldpasswd;
char *newpasswd;
ssize_t llen;
size_t blen;
struct termios term;
tcflag_t saveflag;
printf("User name: ");
fflush(stdout);
blen = 0;
llen = getline(&user, &blen, stdin);
user[llen-1] = 0;
tcgetattr(fileno(stdin), &term);
saveflag = term.c_lflag;
term.c_lflag &= ~ECHO;
tcsetattr(fileno(stdin), TCSANOW, &term);
```

```
    printf("Old password: ");
    fflush(stdout);
    blen = 0;
    llen = getline(&oldpasswd, &blen, stdin);
    oldpasswd[llen-1] = 0;
    printf("\nNew password: ");
    fflush(stdout);
    blen = 0;
    llen = getline(&newpasswd, &blen, stdin);
    newpasswd[llen-1] = 0;
    term.c_lflag = saveflag;
    tcsetattr(fileno(stdin), TCSANOW, &term);
    free(user);
    free(oldpasswd);
    free(newpasswd);
```

APPLICATION USAGE

None.

RATIONALE

Data buffered by the system may make determining the validity of the position of the current file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after fflush() on streams open for read() is not mandated by POSIX.1?2008.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, Standard I/O Streams, fmemopen(), getrlimit(), open_mem? stream(), ulimit()

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