Linux/UNIX System Programming Fundamentals

File I/O

Michael Kerrisk, man7.org © 2026

mtk@man7.org

January 2026

Outline

Rev: # d49d0b54be95

Outline

Files

- "On UNIX, everything is a file"
  - More correctly: "everything is a file descriptor"
- Note: the term **file** can be ambiguous:
  - A **generic term**, covering disk files, directories, sockets, FIFOs, terminals and other devices and so on
  - Or specifically, a **disk file** in a filesystem
  - To clearly distinguish the latter, the term **regular file** is sometimes used

man7.org

System calls versus *stdio*

- C programs usually use *stdio* package for file I/O
- Library functions layered on top of I/O system calls

System calls	Library functions
file descriptor (*int*)	file stream (*FILE **)
open(), *close()*	*fopen()*, *fclose()*
lseek()	*fseek()*, *ftell()*
read()	*fgets()*, *fscanf()*, *fread()* ...
write()	*fputs()*, *fprintf()*, *fwrite()*, ...
–	*feof()*, *ferror()*

- We presume understanding of *stdio*; ⇒ focus on system calls

File descriptors

- All I/O is done using file descriptors (FDs)
  - nonnegative integer that identifies an open file
- Used for all types of files
  - terminals, regular files, pipes, FIFOs, devices, sockets, ...
- 3 FDs are normally available to programs run from shell:
  - (POSIX names are defined in `<unistd.h>`)

FD	Purpose	POSIX name	*stdio* stream
0	Standard input	`STDIN_FILENO`	*stdin*
1	Standard output	`STDOUT_FILENO`	*stdout*
2	Standard error	`STDERR_FILENO`	*stderr*

Key file I/O system calls

Four fundamental calls:

- *open()*: open a file, optionally creating it if needed
  - Returns file descriptor used by remaining calls
- *read()*: input
- *write()*: output
- *close()*: close file descriptor

man7.org

Outline

open(): opening a file

```
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags, ... /* mode_t mode */);
```

- Opens existing file / creates and opens new file
- Arguments:
  - *pathname* identifies file to open
  - *flags* controls semantics of call
    - e.g., open an existing file vs create a new file
  - *mode* specifies permissions when creating new file
- Returns: a file descriptor (nonnegative integer)
  - (Guaranteed to be lowest available FD)

[TLPI §4.3]

open() flags argument

flags is formed by ORing (|) together:

- Access mode
  - Specify exactly one of `O_RDONLY`, `O_WRONLY`, or `O_RDWR`
- File creation flags (bit flags)
- File status flags (bit flags)

[TLPI §4.3.1]

File creation flags

- **File creation flags**:
  - Affect behavior of *open()* call
  - Can't be retrieved or changed
- Examples:
  - `O_CREAT`: create file if it doesn't exist
    - *mode* argument must be specified
    - Without `O_CREAT`, can open only an existing file (else: `ENOENT`)
  - `O_EXCL`: create "exclusively"
    - Give an error (`EEXIST`) if file already exists
    - Only meaningful with `O_CREAT`
  - `O_TRUNC`: truncate existing file to zero length
    - I.e., discard existing file content

man7.org

File status flags

- **File status flags**:
  - Affect semantics of subsequent file I/O
  - Can be retrieved and modified using *fcntl()*
- Examples:
  - `O_APPEND`: always append writes to end of file
  - `O_NONBLOCK`: nonblocking I/O

man7.org

open() examples

- Open existing file for reading:

```
fd = open("script.txt", O_RDONLY);
```

- Open file for read-write, create if necessary, ensure we are creator:

```
fd = open("myfile.txt", O_CREAT | O_EXCL | O_RDWR,
          S_IRUSR | S_IWUSR);       /* rw------- */
```

- Open file for writing, creating if necessary:

```
fd = open("myfile.txt", O_CREAT | O_WRONLY, S_IRUSR);
```

  - File opened for writing, but created with only read permission!

man7.org

read(): reading from a file

```
#include <unistd.h>
ssize_t read(int fd, void *buffer, size_t count);
```

- *fd*: file descriptor
- *buffer*: pointer to buffer to store data
- *count*: number of bytes to read
  - (*buffer* must be at least this big)
  - (*ssize_t* and *size_t* are signed and unsigned integer types)
- Returns:
  - > 0: number of bytes read
    - May be $<$ *count* (e.g., terminal *read()* gets only one line)
  - 0: end of file
  - -1: error
- ⚠ No terminating null byte is placed at end of buffer

man7.org

write(): writing to a file

```
#include <unistd.h>
ssize_t write(int fd, const void *buffer, size_t count);
```

- *fd*: file descriptor
- *buffer*: pointer to data to be written
- *count*: number of bytes to write
- Returns:
  - Number of bytes written
    - May be $<$ *count* (a "partial write")
      (e.g., write fills device, or insufficient space to write entire buffer to nonblocking socket)
  - –1 on error

man7.org

close(): closing a file

```
#include <unistd.h>
int close(int fd);
```

- *fd*: file descriptor
- Returns:
  - 0: success
  - –1: error
- Really should check for error!
  - Accidentally closing same FD twice
    - I.e., detect program logic error
  - Filesystem-specific errors
    - E.g., NFS commit failures may be reported only at *close()*
- **Note**: *close()* **always** releases FD, even on failure return
  - See *close(2)* manual page

man7.org

Example: copy.c

```
$ ./copy old-file new-file
```

- A simple version of *cp(1)*

man7.org

Example: fileio/copy.c

Always remember to handle errors!

```
 1 #define BUF_SIZE 1024
 2 char buf[BUF_SIZE];
 3
 4 int infd = open(argv[1], O_RDONLY);
 5 if (infd == -1) errExit("open %s", argv[1]);
 6
 7 int flags = O_CREAT | O_WRONLY | O_TRUNC;
 8 mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP;     /* rw-r----- */
 9 int outfd = open(argv[2], flags, mode);
10 if (outfd == -1) errExit("open %s", argv[2]);
11
12 ssize_t nread;
13 while ((nread = read(infd, buf, BUF_SIZE)) > 0)
14     if (write(outfd, buf, nread) != nread)
15         fatal("write() returned error or partial write occurred");
16 if (nread == -1) errExit("read");
17
18 if (close(infd) == -1) errExit("close");
19 if (close(outfd) == -1) errExit("close");
```

man7.org

Universality of I/O

- The fundamental I/O system calls work on almost all file types:

```
$ ls > mylist
$ ./copy mylist new         # Regular file

$ ./copy mylist /dev/tty    # Device

$ mkfifo f                  # FIFO
$ cat f &                   # (reads from FIFO)
$ ./copy mylist f           # (writes to FIFO)
```

man7.org

Outline

API summary

```
int open(const char *pathname, int flags, ... /* mode_t mode */);
            // Returns a file descriptor

ssize_t read(int fd, void *buffer, size_t count);
            // Returns: # of bytes actually read or 0 for EOF

ssize_t write(int fd, const void *buffer, size_t count);
            // Returns: # of bytes actually written

int close(int fd);
```

man7.org

Outline

Notes for online practical sessions

- Small groups in **breakout rooms**
  - Write a note into the Discord `#general` channelif you have a preferred group
- **We will go faster, if groups collaborate** on solving the exercise(s)
  - You can **share a screen** in your room
- I will circulate regularly between rooms to answer questions
- Zoom has an "**Ask for help**" button...
- **Keep an eye on the Discord `#general` channel**
  - Perhaps with further info about exercise;
  - Or a note that the exercise merges into a break
- When your room has finished, write a message in the Discord `#general` channel: **"*** Room X has finished ***"**
  - Then I have an idea of how many people have finished

Shared screen etiquette

- It may help your colleagues if you **use a larger than normal font!**
  - In many environments (e.g., *xterm*, *VS Code*), we can adjust the font size with `Control`+`Shift`+"`+`" and `Control`+"`-`"
  - Or (e.g., *emacs*) hold down `Control` key and use mouse wheel
- **Long shell prompts** make reading your shell session difficult
  - Use `PS1='$ '` or `PS1='# '`
- **Low contrast** color themes are difficult to read; change this if you can
- Turn on **line numbering** in your editor
  - In *vim* / *neovim* use: `:set number`
  - In *emacs* use: `M-x display-line-numbers-mode <RETURN>`
    - `M-x` means `Left-Alt`+`x`
- For collaborative editing, **relative line-numbering is evil**....
  - In *vim* / *neovim* use: `:set nornu`
  - In *emacs*, the following should suffice:

```
M-: (display-line-numbers-mode) <RETURN>
M-: (setq display-line-numbers 'absolute) <RETURN>
```

    - `M-:` means `Left-Alt`+`Shift`+`:`

Using *tmate* in in-person practical sessions

In order to share an X-term session with others, do the following:

- Enter the command *tmate* in an X-term, and you'll see the following:

```
$ tmate
...
Connecting to ssh.tmate.io...
Note: clear your terminal before sharing readonly access
web session read only: ...
ssh session read only: ssh S0mErAnD0m5Tr1Ng@lon1.tmate.io
web session: ...
ssh session: ssh S0mEoTheRrAnD0m5Tr1Ng@lon1.tmate.io
```

- Share last "ssh" string with colleague(s) via a text channel
  - Or: "ssh session read only" string gives others read-only access
- Your colleagues should paste that string into an X-term...
- Now, you are sharing an X-term session in which anyone can type
  - Any "mate" can cut the connection to the session with the 3-character sequence `<ENTER> ~ .`
- To see above message again: `tmate show-messages`

man7.org

Exercise notes

- For many exercises, there are templates for the solutions
  - Filenames: `ex.*.c`
  - Look for FIXMEs to see what pieces of code you must add
  - ⚠ You will need to edit the corresponding `Makefile` to add a new target for the executable
    - Look for the `EXERCISE_FILES_EXE` macro

```
-EXERCISE_FILES_EXE = # ex.prog_a ex.prob_b
+EXERCISE_FILES_EXE = ex.prog_a # ex.prog_b
```

- Get a *make* tutorial now if you need one

man7.org

Exercises

1. Using *open()*, *close()*, *read()*, and *write()*, implement the command `tee [-a] file` ([template: `fileio/ex.tee.c`]). This command writes a copy of its standard input to standard output and to `file`. If `file` does not exist, it should be created. If `file` already exists, it should be truncated to zero length (`O_TRUNC`). The program should support the `-a` option, which appends (`O_APPEND`) output to the file if it already exists, rather than truncating the file.

   Some hints:

   - You can build `../libtlpi.a` by doing *make* in source code root directory.
   - Standard input & output are automatically opened for a process.
   - Remember that you will need to add a target in the `Makefile`!
   - After first doing some simple command-line testing, test using the unit test in the `Makefile`: `make tee_test`.
   - Why does "`man open`" show the wrong manual page? It finds a page in the wrong section first. Try "`man 2 open`" instead.
   - `while inotifywait -q . ; do echo -e '\n\n'; make; done`
     - You may need to install the *inotify-tools* package
   - Command-line options can be parsed using *getopt(3)*.

This page intentionally blank