Reading 8

✱ I/O in C

- Every running programs begins with three default I/O streams
  - 0: stdin
  - 1: stdout
  - 2: stderr

- Different placeholders for printf():

```c
%f, %g: placeholders for a float or double value
%d:     placeholder for a decimal value (char, short, int)
%u:     placeholder for an unsigned decimal
%c:     placeholder for a single character
%s:     placeholder for a string value
%p:     placeholder to print an address value

%ld:    placeholder for a long value
%lu:    placeholder for an unsigned long value
%lld:   placeholder for a long long value
%llu:   placeholder for an unsigned long long  value
```

```
%x:     print value in hexadecimal (base 16)
%o:     print value in octal (base 8)
%d:     print value in signed decimal  (base 10)
%u:     print value in unsigned decimal (unsigned base 10)
%e:     print float or double in scientific notation
(there is no formatting option to display a value in binary)
```

- A file stores persistant data

- Remember file pointers:

```
FILE *infile;
infile = fopen("input.txt", "r");
if (infile == NULL){
    printf("Error");
    exit(1);
}
```

- Remember to close files via fclose().

- `rewind(infile)` resets current position to beginning of file
- `fseek()` allows you to move to a specific location in the file:
  - ↳ `fseek(FILE *f, long offset, int whence);`
    - ↳ `fseek(f, 3, SEEK_CUR);` → seek 3 chars forward from current position
    - ↳ `fseek(f, -3, SEEK_END);` → seek 3 chars back from the end of the file

- fscanf() placeholders:

```
%d integer
%f float
%lf double
%c character
%s string, up to first white space

%[...] string, up to first character not in brackets
%[0123456789] would read in digits
%[^...] string, up to first character in brackets
%[^\n] would read everything up to a newline
```

Similar to regex, can be handy!!!

✡ Files and Directories:

- A file is simply a linear array of bytes, each of which you can read or write.
  - ↳ the low-level name of a file is often referred to as its inode number (i-number).

- Extensions such as '.txt' or '.c' are convention, there is usually no enforcement that the data contained in a file named 'main.c' is indeed C source code.

- System call to create file "foo" in current directory:

```
int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC, S_IRUSR | S_IWUSR);
```

combining flags via bitwise or

O_CREAT → create file if it doesn't exist

O_WRONLY → open in write only mode

O_TRUNC → erase contents if file already exists

- open() returns a file descriptor.
  - ↳ an integer, private per process, used to access files
  - ↳ you can also think of it as a pointer to an object of type file
    - ↳ then you can call other "methods" like 'read()' or 'write()'

- A simple array indexed by the file descriptor tracks which files are opened on a per-process basis. Each entry of the array is just a pointer to 'struct file'

- use strace to track all the system calls a process makes.
  - ↳ it has some handy arguments (check man page).

- Ex using strace

```
prompt> strace cat foo
...
open("foo", O_RDONLY|O_LARGEFILE)          = 3
read(3, "hello\n", 4096)                   = 6
write(1, "hello\n", 6)                     = 6
hello
read(3, "", 4096)                          = 0
close(3)                                   = 0
...
prompt>
```

file descriptor
opening (read only)
size of buffer
# of bytes read/wrote
read results
reading
indicates EOF

- Use lseek() system call to get to a particular location using an offset.

- Parent processes can create children processes via fork() system call.

- the dup() call allows a process to create a new file descriptor that refers to the same underlying open file as an existing descriptor.

- when you use write() there is a smallish delay before data is actually put into persistent data.
  - ↳ you can use fsynch() after write() to update the persistant data instantaneously.

- mv command makes a system call to rename().
  - ↳ rename() is an atomic call meaning if the system crashes during the renaming, the file will either be named the old name or new name, no odd in-between state can arise.

- To access metadata we can make the stat() or fstat() system calls.

- rm calls ulink() to remove a file.

- mkdir makes a call to mkdir()
  - ↳ although a newly created directory is "empty", it has 2 entries:
    1. entry that refers to itself
    2. entry that refers to its parents

- When you create a file you do 2 things:
  1. making a structure (the inode) that will track virtually any relevant info about the file
  2. linking a human-readable name to the file, and putting that link into a directory.

- Symbolic links are formed by holding the pathname of the linked-to file as the data of the link file.

- Hard links refer to the same inode number of the original file.

- In `ls -l` output if the first character is "l" that signifies a Symbolic link

• Summary

ASIDE: KEY FILE SYSTEM TERMS

- A **file** is an array of bytes which can be created, read, written, and deleted. It has a low-level name (i.e., a number) that refers to it uniquely. The low-level name is often called an **i-number**.
- A **directory** is a collection of tuples, each of which contains a human-readable name and low-level name to which it maps. Each entry refers either to another directory or to a file. Each directory also has a low-level name (i-number) itself. A directory always has two special entries: the `.` entry, which refers to itself, and the `..` entry, which refers to its parent.
- A **directory tree** or **directory hierarchy** organizes all files and directories into a large tree, starting at the **root**.
- To access a file, a process must use a system call (usually, `open()`) to request permission from the operating system. If permission is granted, the OS returns a **file descriptor**, which can then be used for read or write access, as permissions and intent allow.
- Each file descriptor is a private, per-process entity, which refers to an entry in the **open file table**. The entry therein tracks which file this access refers to, the **current offset** of the file (i.e., which part of the file the next read or write will access), and other relevant information.
- Calls to `read()` and `write()` naturally update the current offset; otherwise, processes can use `lseek()` to change its value, enabling random access to different parts of the file.
- To force updates to persistent media, a process must use `fsync()` or related calls. However, doing so correctly while maintaining high performance is challenging [P+14], so think carefully when doing so.
- To have multiple human-readable names in the file system refer to the same underlying file, use **hard links** or **symbolic links**. Each is useful in different circumstances, so consider their strengths and weaknesses before usage. And remember, deleting a file is just performing that one last `unlink()` of it from the directory hierarchy.
- Most file systems have mechanisms to enable and disable sharing. A rudimentary form of such controls are provided by **permissions bits**; more sophisticated **access control lists** allow for more precise control over exactly who can access and manipulate information.