Today

- Learning Outcomes
  - Define
    - Supervisor mode
    - User mode
    - Processor privilege level
    - System call
    - File descriptor
    - File permissions (mode)
  - Use the system calls open, close, read, write.
- Reading
  - Chapter 10: 10.1-10.4

Processor Privilege levels

- Modern hardware has multiple privilege levels.
- Different software can run with different privileges.
- Processor hardware typically provides (at least) two different modes of operation:
  - User mode: how all "regular programs" run.
  - **Kernel mode or supervisor mode: how the OS runs.**
  - Most processors have only two modes; x86 has four; some older machines had 8!
- The mode in which a piece of software is running determines:
  - What instructions may be executed.
  - What memory locations may be accessed (enforced through translation).
  - Whether the program can directly interface with devices.

System Calls

- When a program wants to do something that it cannot do in user mode (e.g., interact with a disk), it needs to ask the kernel to perform the activity on its behalf.
- System calls are the mechanism that accomplishes this.
  - They are a way to for a program to explicitly request the operating system to do something.
  - System calls appear in section 2 of the man pages.

The POSIX File System API

- The file system APIs are implemented as library functions (in `libc`) that invoke system calls.
- `open`: takes a pathname (the name of a file) and returns a file descriptor (`fd`), which you will use to access that file.
- `close`: takes an `fd` and frees up all the OS and library resource that have been allocated to it.
- `read`: reads data from the file referenced by `fd` into a user-specified buffer.
- `write`: takes data from a buffer and writes it to the file referenced by `fd`.

A Copy Program in Pseudo code

```
copy(infile, outfile)
   fd_in = open infile for reading
   fd_out = open outfile for writing
   while (there is data left to read)
      read data from fd_in
      write data to fd_out
   close fd_in
   close fd_out
```

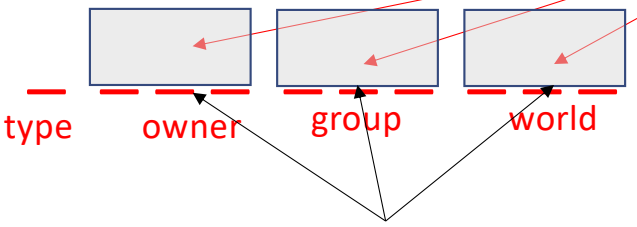
open

- `int open(const char *path, int oflag, int mode)`
- Parameters:
  - `path`: the path name of the file you wish to open
  - `oflag`: flag values that tell you things such as:
    - How you want to access the file: `O_RDONLY, O_WRONLY, O_RDWR`
    - What to do if the file does/does not exist (`O_CREATE, O_EXCL`
    - How the file should behave with respect to (OS) caching (`O_SYNC, O_DIRECT`).
  - `mode`: if the file does not exist and you create it, what should its access mode be set to (e.g., who is allowed to read/write it). You can ignore this for now; we'll come back to it.
- Return value: a file descriptor
  - non-negative integer on success; -1 on error).
- Example
  - `open("myfile", O_CREAT, 0600)`

A Brief Digression on File System Modes

```
mseltzer@valdes:~$ ls -asl
total 1592
 40 drwx------   5 mseltzer instruct   448 Oct 27 19:28 ./
280 drwxr-xr-x 952 root     root     23819 Oct 28 17:23 ../
 40 drwx------   8 mseltzer instruct   167 Sep 21 10:53 313/
192 -rw-------   1 mseltzer instruct 36600 Mar 29  2020 foo
 40 drwx------   2 mseltzer instruct    26 Apr 19  2018 Mail/
 64 -rw-r--r--   1 mseltzer instruct   725 Sep 14 17:08 xxx.c
```

Permissions or access modes



Read
Write
eXecute

- regular file
d directory
l link

We typically specify permissions using 3 octal digits corresponding to the permissions for each category of user.

close

- `int close(int fd)`
- Parameters:
  - `fd`: the file descriptor to be closed
- Return value: 0 on success; -1 on error
- Example
  - `int ret = close(fd);`

read

Note: There was some voiceover in the video that might be confusing:
A return value of 0 is the **only return that always indicates end of file**.
A short read *may* indicate end of file but does not necessarily do so.
Therefore, after a short read, you should try to read again.

- `ssize_t read(int fd, void *buf, size_t nbyte)`
- Parameters:
  - `fd`: file descriptor for the file from which to read
  - `buf`: a buffer into which the bytes read are placed
  - `nbyte`: the number of bytes to read
- Return value: a `ssize_t` (a `signed long`)
  - On success: the number of bytes read
  - On end-of-file: 0
  - On failure: -1
- Example
  - `char buf[4096];`
  - `ssize_t bytes_read = read(fd, buf, 4096);`

write

- `ssize_t write(int fd, const void *buf, size_t nbyte)`
- Parameters:
  - `fd`: file descriptor for the file to which to write
  - `buf`: a buffer containing the data to write
  - `nbyte`: the number of bytes to write
- Return value: a `ssize_t` (a `signed long`)
  - On success: the number of bytes written
  - On failure: -1
- Example
  - `ssize_t bytes_written = write(fd, buf, 4096);`