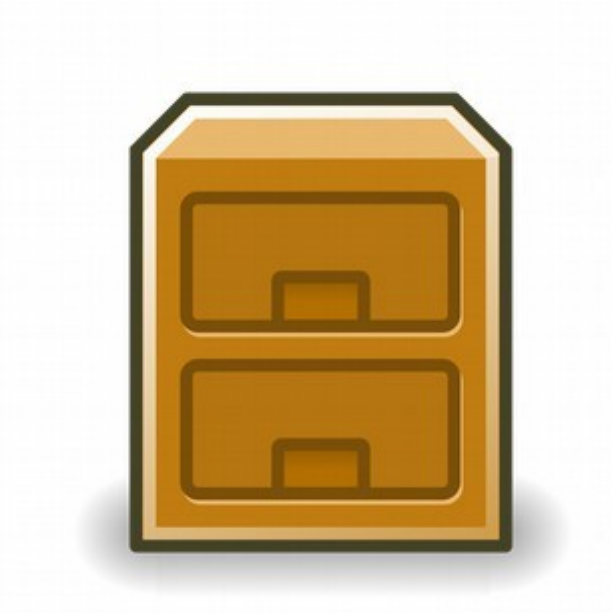
CS 261
Fall 2021

Mike Lam, Professor



Files

Files

- A file is a sequence of bytes
  - Logical abstraction provided by the operating system
  - In Linux, many things are represented as files
    - All I/O is performed by reading/writing "files"
  - Raw format on disk is determined by file system
    - Common file systems: FAT32, NTFS, HFS+, ext4, Lustre
- Basic file operations:
  - Open a file (returns a file descriptor integer identifier)
  - Change current position (seek)
  - Read and write bytes
  - Close a file (kernel does this if the process does not)

Files

- Regular files – contain arbitrary data
  - Binary vs. text file distinction (applications only)
  - Context is crucial! (*Info* = *Bits* + *Context*)
    - All files are “binary”!
- Directory files – contain links to other files
  - Special links: "." (self) and ".." (parent)
- Socket files – links to another process
  - Could be on another computer
  - Used for inter-process communication (IPC)
  - You'll learn to use these in CS 361

Files

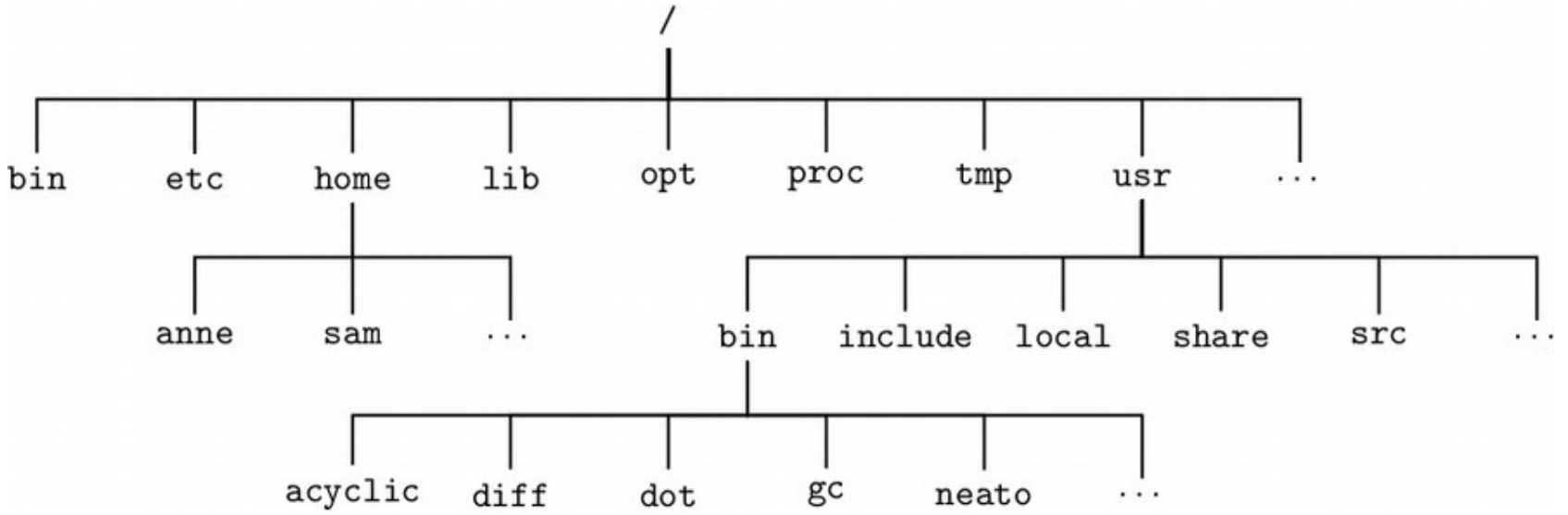
- Pipes - link between two processes
  - Persist as long as the processes are running
  - Named pipes persist outside of any processes
- Symbolic (“soft”) links - contains a reference to another file
  - A hard link (not a file!) is just a pointer to a shared inode
- Character/block devices - access to hardware
  - Unbuffered (character) or buffered (block)
  - Examples: hard disks, keyboard, printers, terminals
- Pseudo-devices - utilities provided by OS
  - `/dev/null` - discards input; no output
  - `/dev/zero` - outputs continuous stream of zero bytes
  - `/dev/random` and `/dev/urandom` - outputs pseudo-random numbers

File systems

- File systems abstract the details of file storage
  - Manage logical → hardware mapping
  - Manage metadata (stored in inodes)
- File systems must be mounted
  - One “root” file system (“/”); use `mount` to add others
  - Mounted into a specific mount point in root file system
  - Usually auto-mounted according to `/etc/fstab`
  - Use `df` utility to view mounted file systems
  - File system can be mounted from another machine
    - Networked File System (NFS)

File system hierarchy

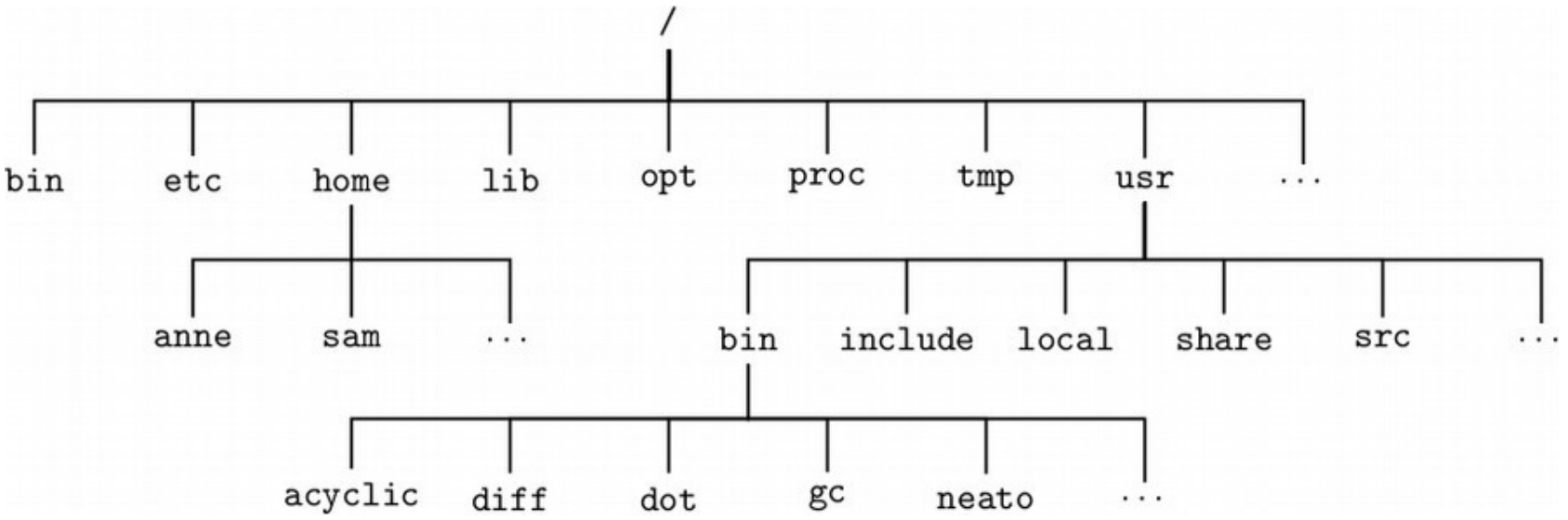
- File system hierarchy standard (FHS)
  - Standard layout of files on a Linux system
- Absolute vs. relative pathnames
  - Absolute: path from root (/)
  - Relative: path from current working directory



Question

- What is the absolute pathname for the “sam” folder?

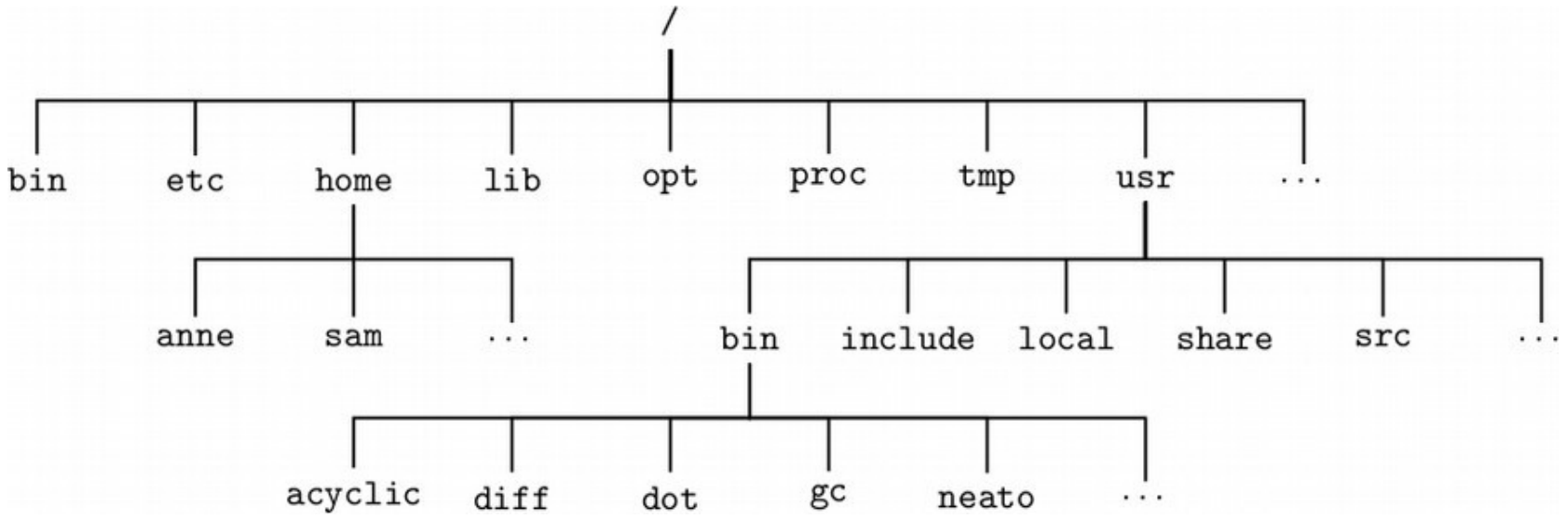
/home/sam



Question

- Assume you are in the “anne” folder. What is the relative pathname for the “sam” folder?

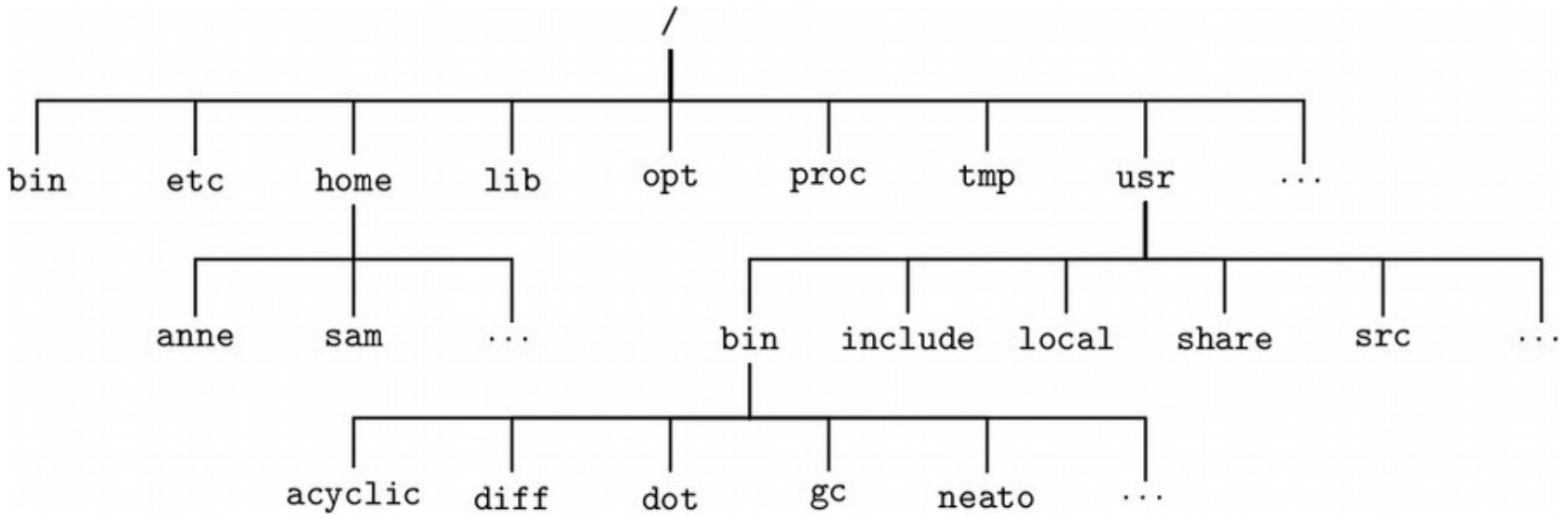
```
../sam
```



Question

- What is the absolute pathname for the “dot” utility?

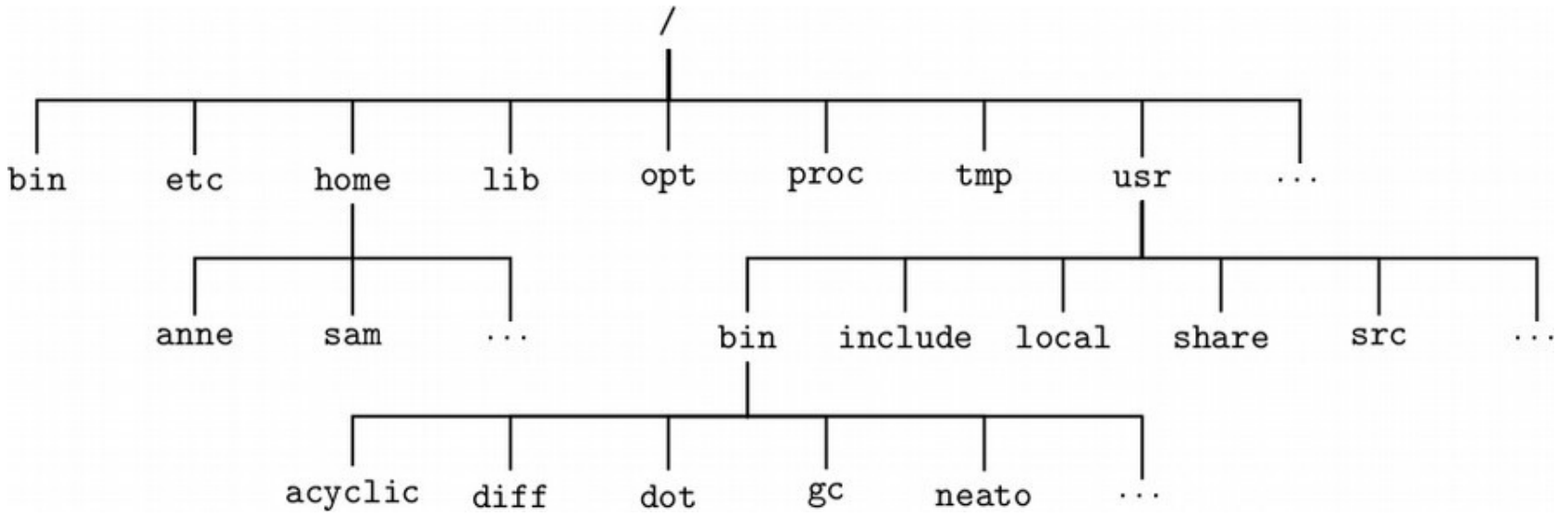
/usr/bin/dot



Question

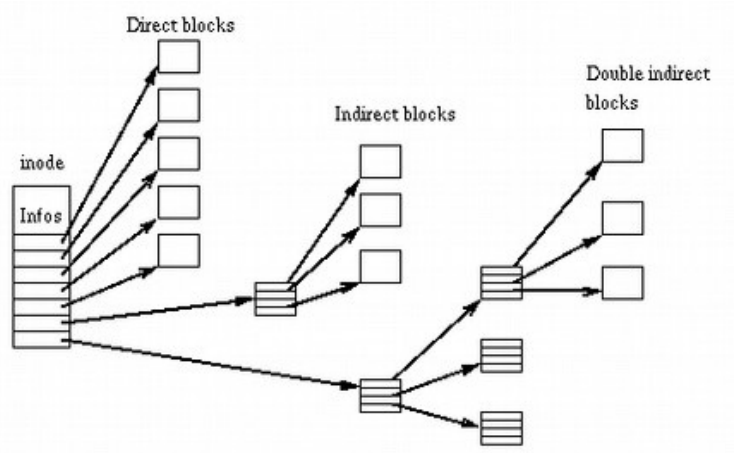
- Assume you are in the “anne” folder. What is the relative pathname for the “dot” utility?

```
../../usr/bin/dot
```



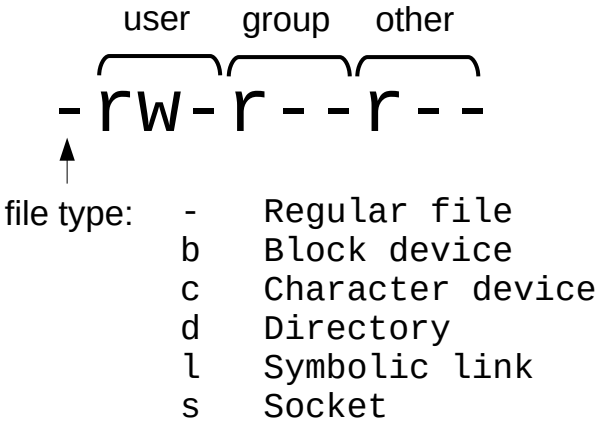
File metadata

- Metadata is information about a file
  - Stored in an inode by the file system or kernel
  - Use `stat()` or `fstat()` to obtain a file's metadata
  - Need `unistd.h` and `sys/stat.h`
  - Information:
    - File type (regular, directory, socket)
    - User and group owner IDs
    - Access permissions
    - Total size (in bytes or blocks)
    - Date/time of last access/modification
    - Device ID
    - Pointers to file data on device (direct or indirect)



File permissions

- Traditional Unix permissions
  - Three bits: read, write, execute
    - Stored in inode; interpreted using octal
  - Three categories: user, group, other
  - Every file has a user owner and a group
    - "Other" = everyone else (not owner or in group)
  - See output of "`ls -l`" and "`groups`"
  - Change permissions using `chmod`
    - `chmod u+x <file>` *(add execute permission for user)*
    - `chmod go-w <file>` *(remove write permission for group/other)*
    - `chmod a+r <file>` *(add read permission for everyone)*
    - `chmod 644 <file>` *(set permissions to `rw-r--r--`)*



Question

- Give the Unix permissions in octal of a file that can be read, written, and executed by anyone (not a fantastic idea from a security standpoint!).

```
777    -rwxrwxrwx
```

Question

- Give the Unix permissions in octal of a file that can be read and executed by anyone but only read, executed, and written by the owner (e.g., a compiled program).

```
755   -rwxr-xr-x
```

Question

- Give the Unix permissions in octal of a file that can be read only by the owner and not written or executed by anyone (e.g., an SSH key file).

```
400  -r--------
```

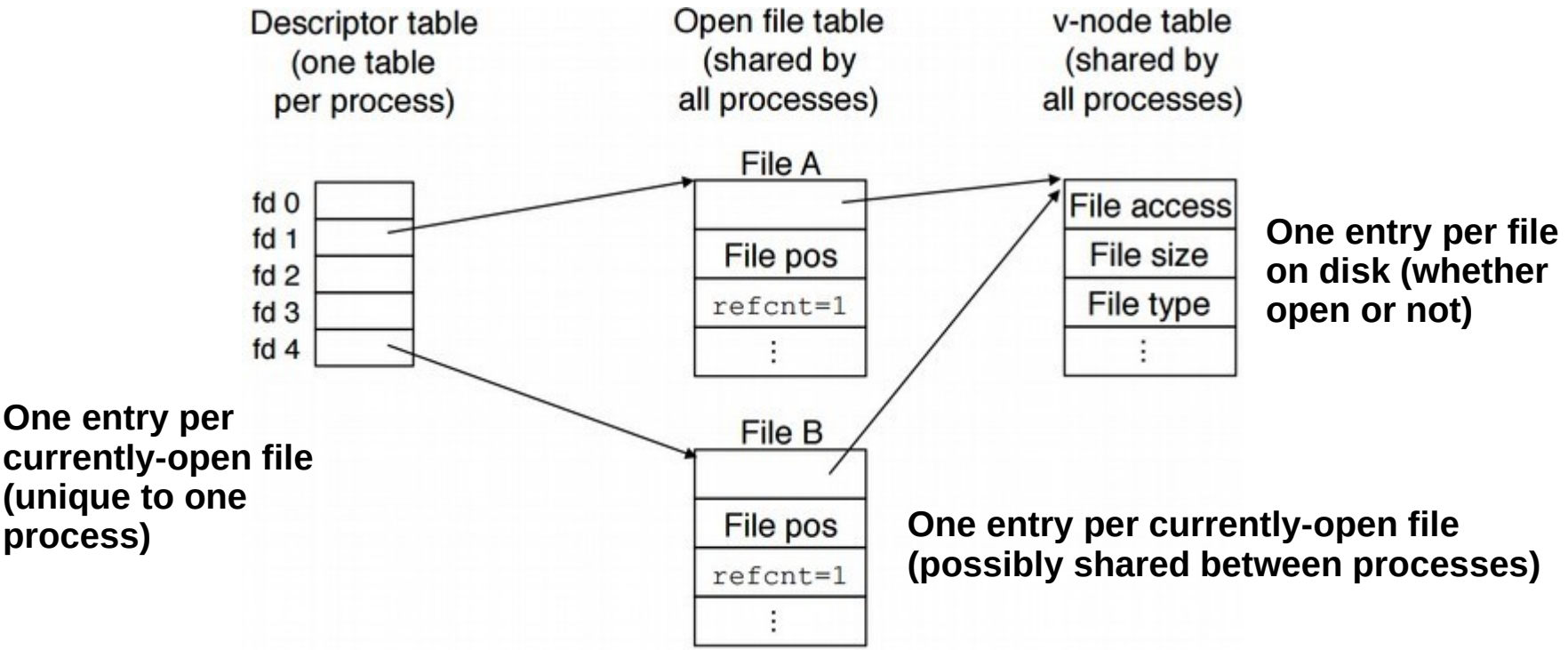
File permissions

- Access Control Lists (ACLs)
  - Newer mechanism (more complex but more flexible)
  - Any desired permission at any desired granularity
    - `getfacl()` / `setfacl()`
  - Useful for fine-grained permissions
    - Example: your PA submission folders for this class
  - Interactions with traditional permissions can be tricky
    - Effective permissions are the intersection of traditional and ACL

```
user:lam2mo:rwx          // sample permissions for
user:weikleda:rwx        // CS 261 submissions
user:<YOUR_EID>:rwx
group:csmajor:---
other::---
```

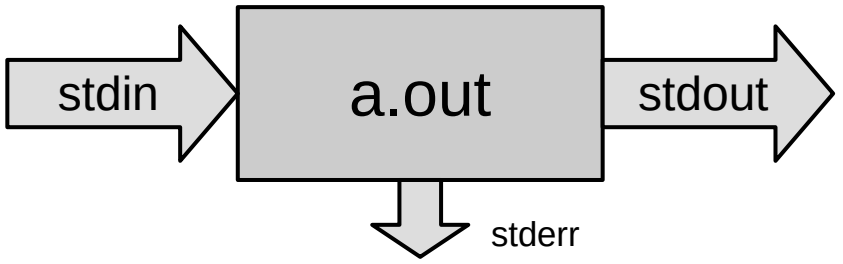
File sharing

- Open files can be shared among processes via OS
  - Descriptor tables (per-process) - duplicated on fork
  - Open file table (shared) - use `lsof` utility to view
  - inode table (shared) - called “v-node” table in textbook



Standard I/O

- Three C standard file descriptors for every process
  - Standard input (`stdin`) (0)
  - Standard output (`stdout`) (1)
  - Standard error (`stderr`) (2)
  - In Java: `System.in`, `System.out`, and `System.err`
- Used by default in some places
  - `printf("Hello!")` means `fprintf(stdout, "Hello!")`

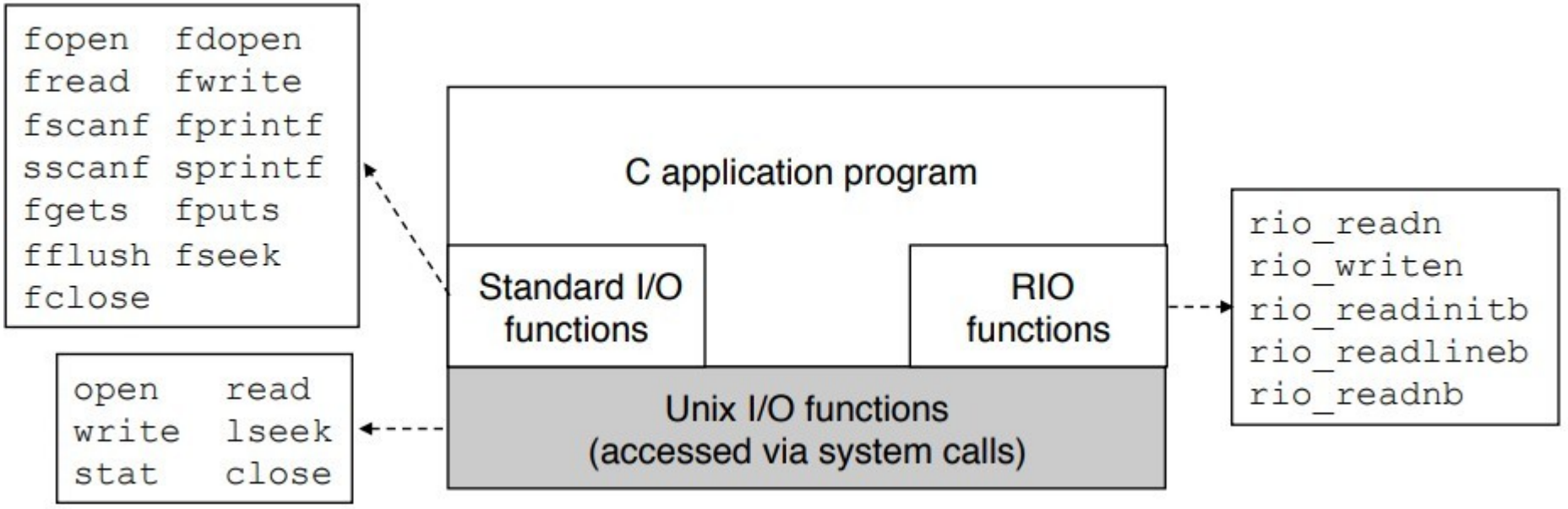


File I/O functions

- **Unix I/O** functions
  - `open, read, write, lseek, stat, close`
  - Thin wrappers for system calls
  - Uses integer file descriptors
- C **standard I/O** functions (libc)
  - `fopen, fread, fgets, fwrite, fprintf, fseek, fclose`
  - Provides buffering and line ending translation
  - Uses `FILE*` file stream abstraction around file descriptors
  - More portable!
- Textbook's **robust I/O** routines
  - Wrappers for buffered terminal/socket I/O (no short counts)
  - We won't use them in this course

File I/O functions

- General guidelines (from textbook)
  - Use the standard I/O functions whenever possible
  - Don't use `scanf` to read binary files
  - Use the robust I/O functions for network sockets



I/O redirection

- Linux shells allow you to redirect standard I/O streams
  - Standard out: `echo "Hello" > data.txt`
    - By default, prints to the console
  - Standard in: `wc < data.txt`
    - By default, reads from the keyboard
    - Use **CTRL-D** to signal "end" of input
  - Standard err: `./mybigapp 2> log.txt`
  - Out and err: `./mybigapp &> output.txt`
  - Pipes: `ls */*.c | grep "p4"`
    - Can combine with redirection: `ls */*.c | grep "p4" > p4-files.txt`

← Useful for testing `iotrap` in P4! (put the input in a file and redirect it to stdin)

System design

- Unix system design philosophy:
  - Write programs that do one thing and do it well
  - Write programs to work together
  - Write programs to handle text streams, because that is a universal interface

Example:

Determine the ten most-frequently-used words in the complete works of William Shakespeare.

```
curl https://www.gutenberg.org/files/100/100-0.txt |
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c |
sort -rn | sed 10q
```

Question

- How many processes will the following command create?

```
curl https://www.gutenberg.org/files/100/100-0.txt |
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c |
sort -rn | sed 10q
```

OS Themes

- Information = Bits + Context
- Abstraction helps manage complexity
- Systems software is a foundation