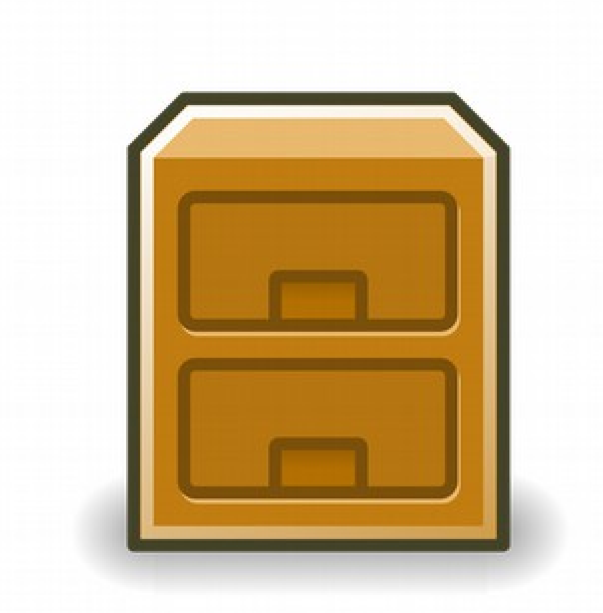
CS 261
Fall 2017

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

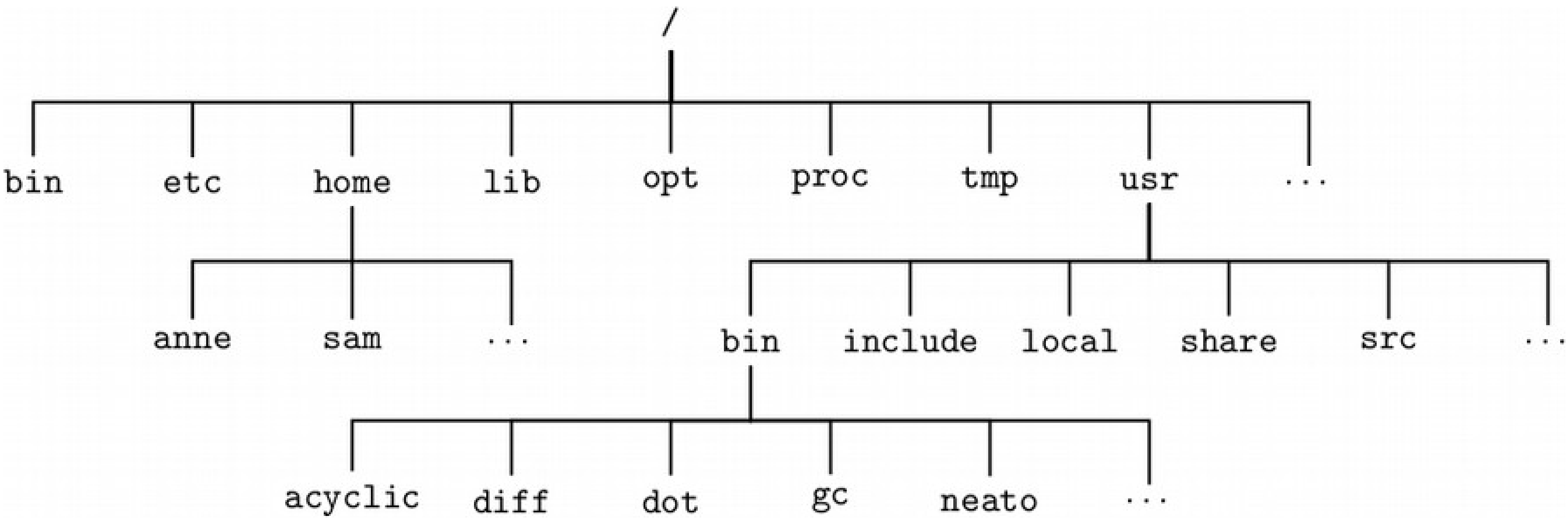
- Regular files – contain arbitrary data
  - Binary vs. text file distinction (applications only)
  - Context is crucial! (*Info* = *Bits* + *Context*)
    - All files are “binary”!
- Directory files – contain links to other files
  - Special links: "." (self) and ".." (parent)
- Socket files – links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

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)
- Absolute vs. relative pathnames
  - Absolute: path from root (/)
  - Relative: path from current working directory



Directory contents

- Use "dirent" functions and DIR* abstraction
  - #include dirent.h
  - opendir(), dirent(), closedir()

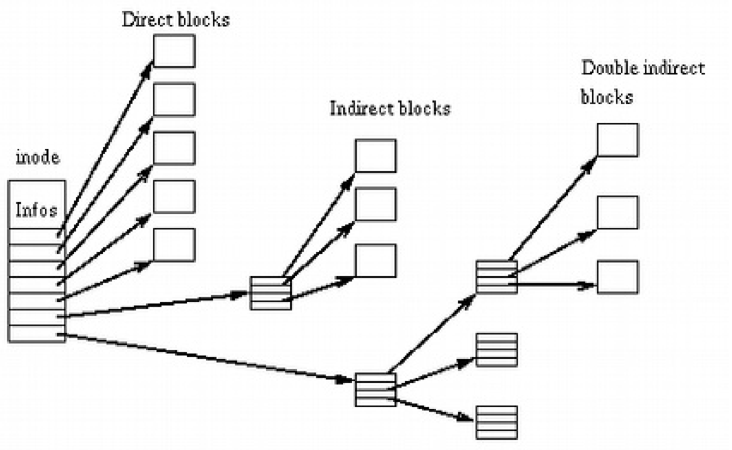
```
// open current directory listing
DIR *dir = opendir(".");
struct dirent *entry = readdir(dir);
while (entry != NULL) {

    // print file information
    printf("[%d] %s (%d bytes)\n",
            (int)entry->d_ino, entry->d_name);

    // next file in listing
    entry = readdir(dir);
}
closedir(dir);
```

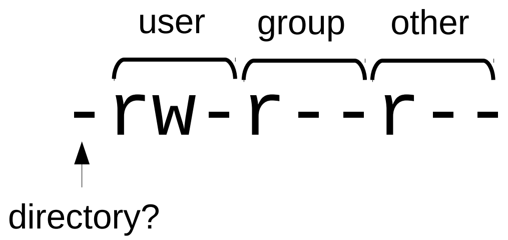
File metadata

- 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--`)*

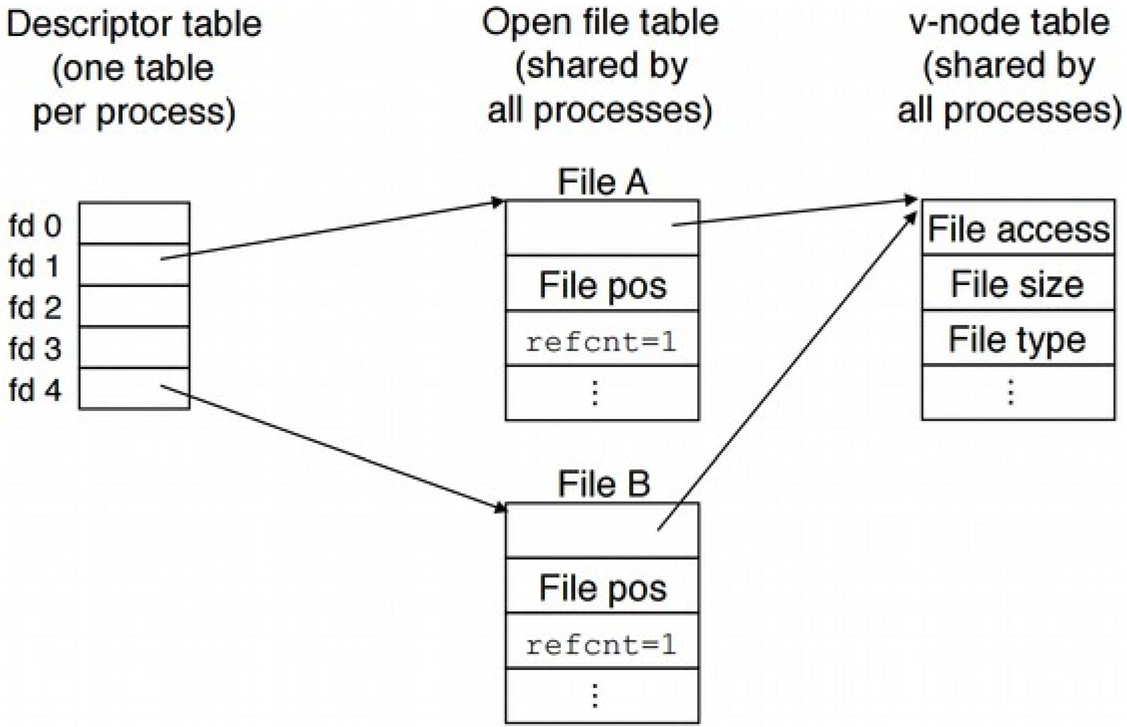


File permissions

- 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

File sharing

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

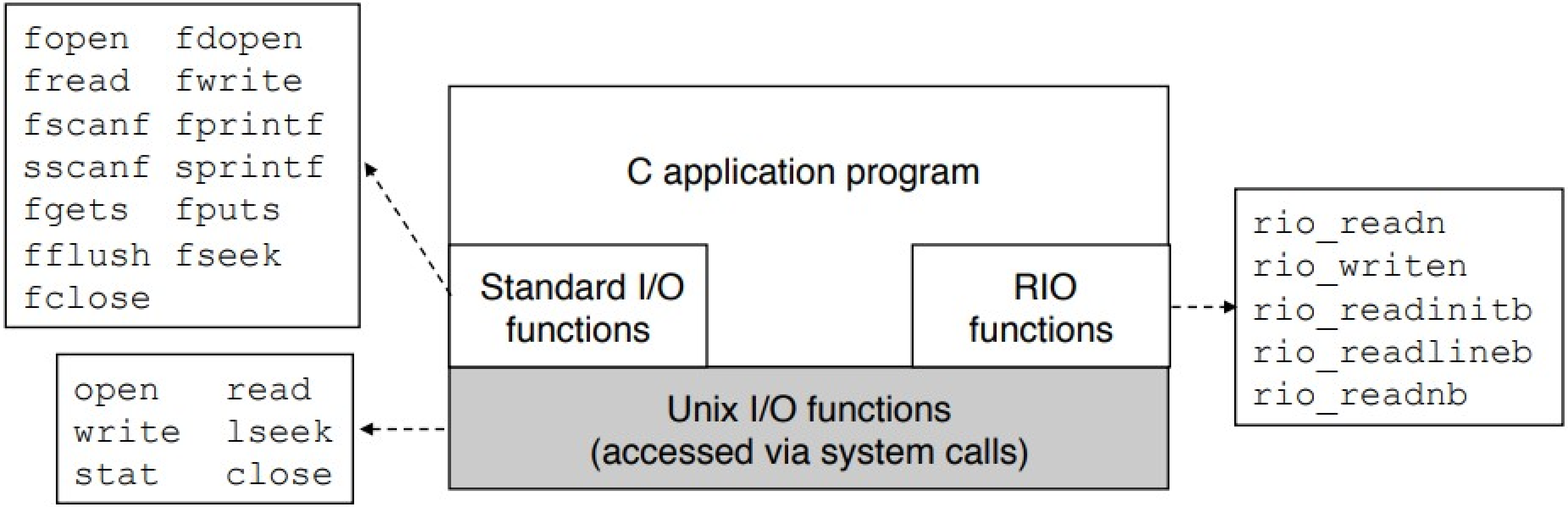


File I/O functions

- **Unix I/O** functions
  - `open, read, write`
  - 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



fopen fdopen
fread fwrite
fscanf fprintf
sscanf sprintf
fgets fputs
fflush fseek
fclose

open read
write lseek
stat close

C application program

Standard I/O functions

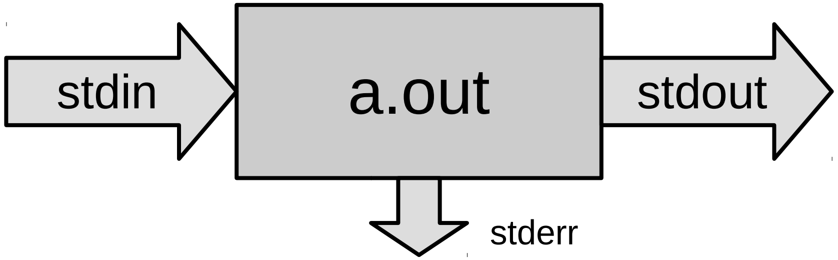
RIO functions

Unix I/O functions
(accessed via system calls)

rio_readn
rio_writen
rio_readinitb
rio_readlineb
rio_readnb

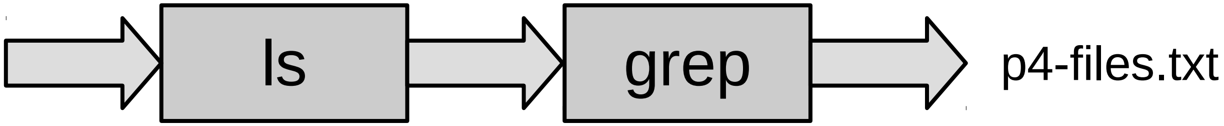
Standard I/O

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



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`



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

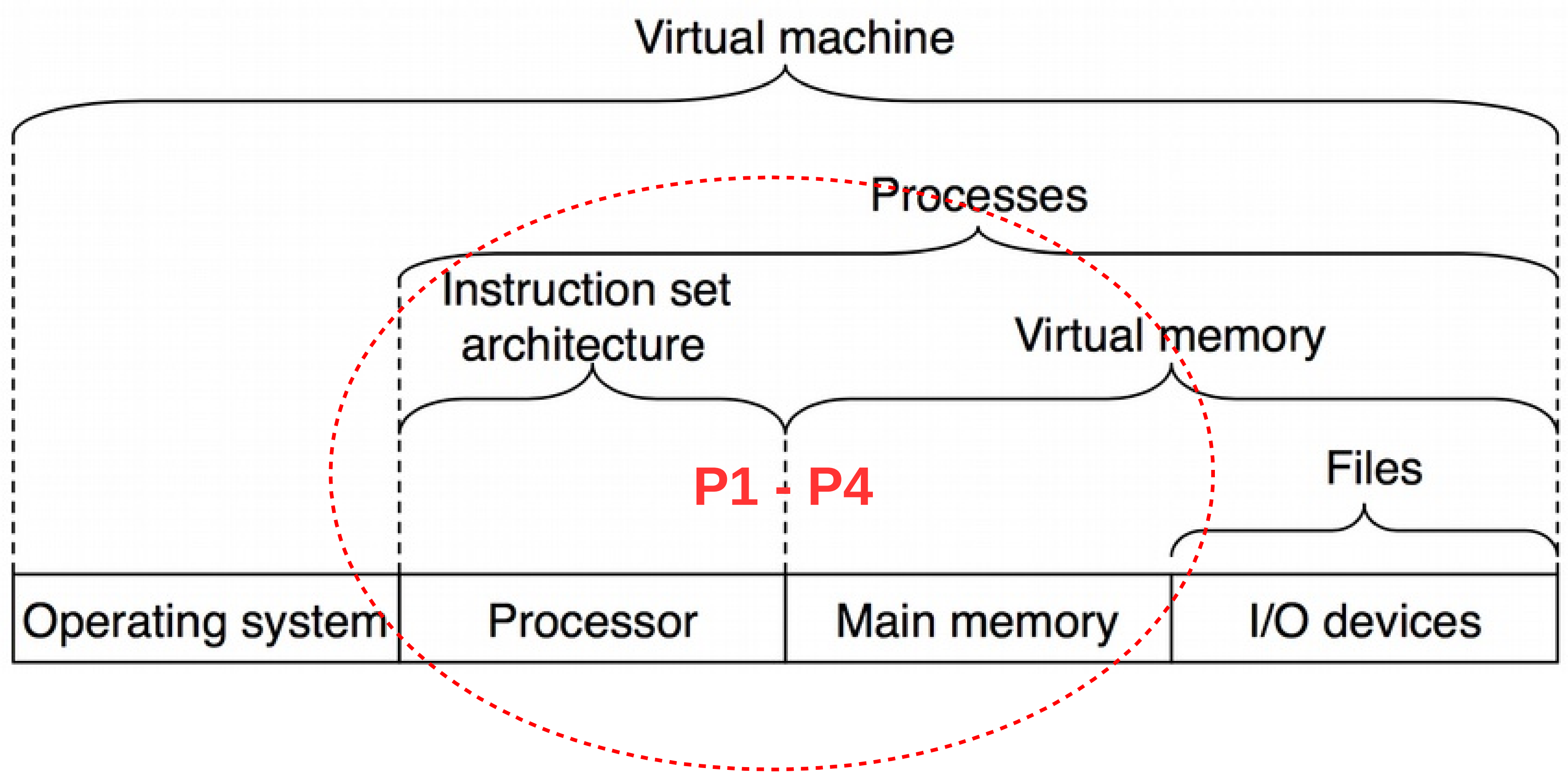
Example:

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

```
curl http://www.gutenberg.org/files/100/100.txt |
tr -cs A-Za-z '\n' | tr A-Z a-z | sort | uniq -c |
sort -rn | sed 1q
```

Review: Operating Systems

- Bits + Context
- Abstraction


Virtual machine
Processes
Instruction set architecture
Virtual memory
Files
P1 - P4
Operating system
Processor
Main memory
I/O devices