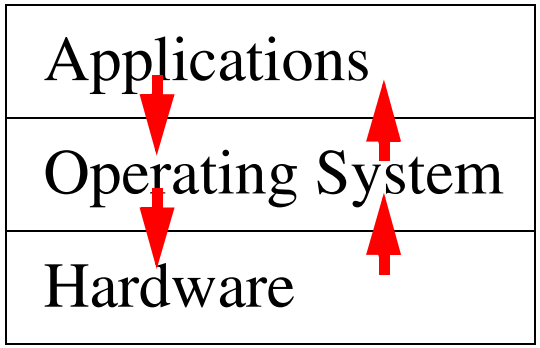
CS 126 Lecture P2: Introduction to Unix

Outline

- **Background**
- Files
- Processes
- Interactions
- Conclusion

Operating Systems



- What does an OS do?
  - Make lives easy: hides low level details of bare machine
  - Make lives fair: arbitrate competing resource demands
- What we learn here: the interfaces by OS to upper layer
  - User interface
  - Programmer's interface
  - Command line vs. graphical user interface (more later)

Operating systems

Multics

timesharing
file system, protection
virtual machines

OS/360

Macintosh

windows, mouse

DOS

PC standard OS

UNIX/Linux

- C language, bootstrapped implementation
- integrated command structure
- simplified, integrated file system
- used by most programmers

Windows

[OS definition under litigation]

A Brief History

- Multics (65-70)
  - Ambitious OS project at MIT
  - Pioneered most of the innovations in modern OS
  - A little ahead of its time
- Unix
  - Thompson and Ritchie (69): simplicity and elegance
  - AT&T (70-80s):
    continued development and "shepherding" it out of AT&T
  - Berkeley ("BSD") (78-93): maturation (e.g. TCP/IP)
  - Various flavors of commercial Unix (80-90s):
    convergence and fragmentation
  - Linux (91-): new life

Outline

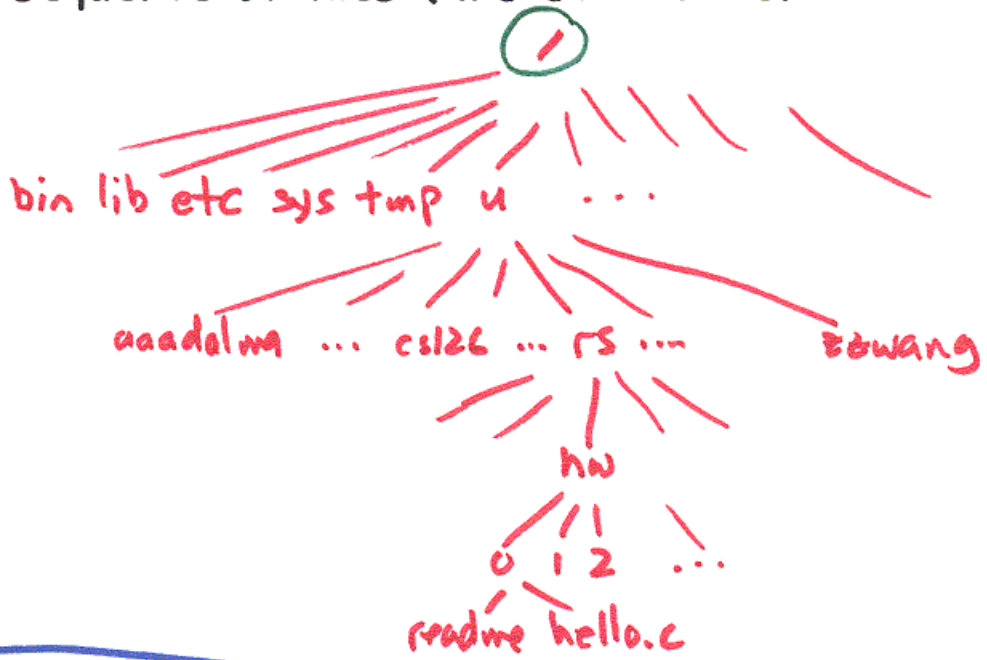
- ~~Background~~
- **Files**
  - A simple and powerful abstraction for storage (disks)
  - Extended for things beyond disks
- Processes
- Interactions
- Conclusion

File System

- "Everything in UNIX is a file"
- Abstract mechanism for "permanent" storage

file:
sequence of bytes

directory: "/" **(like folders)**
sequence of files (and directories)



filename:
sequence of directory names
on the path from "/" to the file

A Hierarchical Name Space: Same as folders and files on Windows or MacOS

File manipulation commands

cat, more	show the contents
cp	copy
rm	remove (delete)
mv	move (rename)
ls	list file names
mkdir, rmdir	create, delete directory
pwd	name of current directory
cd	change directory

"." current directory
".." parent directory
"~" my home directory
"~xx" xx's home directory

chmod	change permissions mode

"*" any sequence of characters

DON'T TYPE "rm *"

Outline

- ~~Background~~
- ~~Files~~
- **Processes**
  - An abstraction for the processor (CPU)
  - "Everything" (almost every command) is a process
- Interactions
- Conclusion

Some Unix commands

lpr	output to printer
man, apropos	online documentation
grep, awk, sed	pattern search (stay tuned)
sort	sort the lines
diff	show differences
cal, date, time	time utilities
mail, news, pine	communication
bc, dc	calculators
cc, lcc, gcc	C compile
gs, xv	view graphics
history	past commands typed

- Over 2500 "standard" commands
- Thousands more "available" programs

emacs, tex, latex	text in and out
netscape	access web

- A Unix "command" is the same as a Windows "program"
- Instead of clicking its icon under Windows, we simply type its name to invoke it on a command line.

Multiprocessing

- Abstraction provided by operating system
  - multiple "virtual" machines for your use
  - outgrowth of 1960s "time-sharing"
  - not found on 1st-generation PC OS's

Multiple windows "active"??

Ex:

```
emacs hello.c &
```

ampersand indicates "do this in the background"
alternatively, could use ctrl-z (and bg)

```
% emacs hello.c &
[1] 18439
% netscape &
[2] 18434
% jobs
[1]   + Running                 emacs hello.c
[2]   - Running                 netscape
%
```

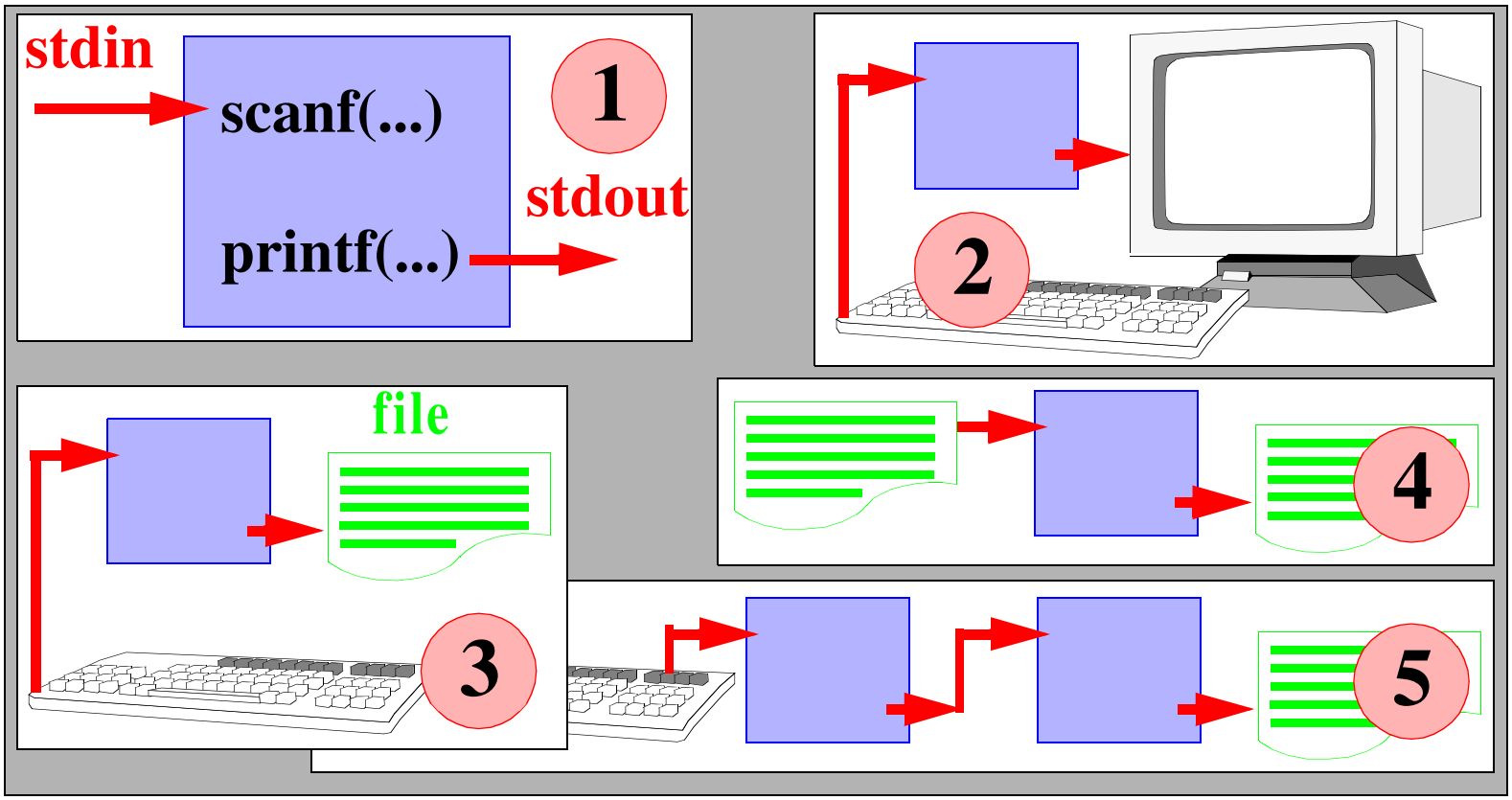
For COS126

- one window for editor
- one window for UNIX commands
  lcc, a.out, ls, cp
  [one window for output]

Outline

- ~~Background~~
- ~~Files~~
- ~~Processes~~
- **Interactions**
  - (between files and processes)
- Conclusion

I/O Redirection and Pipes



- 1: “Standard I/O”, 2: default attachment, 3: redirect output
- 4: redirect both input and output, 5: pipes

Filters and pipes

- Standard Input, Standard Output

- abstract files for command interfaces

Redirection:

- standard input from file
- standard output to file

```
a.out > saveanswer
sort < myfile > myfilesorted
```

3 on prev. slide

4 on prev. slide

Piping:

- connect standard output of one command to standard input of the next

```
ls | wc -l > outputfile
plotprog | lpr
gamblerall | avg
```

5 on prev. slide

Don't confuse redirection and piping

```
plotprog > lpr
```

C Shell (/bin/csh)

```
#!/bin/csh -f
printf "Hello world! Give me a number:\n"
set n = $<
printf "Thanks! I've always been fond of %d\n" $n
```

Don't worry about the details here.

- The program that's running inside your terminal window
- Much more than just manipulating files and launching commands
- It's an "interpreter", with its own powerful programming language!
- Try your first "csh script"?

Shell

Command interface to UNIX

- Just another programming language
  - sequences of instructions

```
mv file1 tmp; mv file2 file1; mv tmp file
```

  - variables

```
printenv
```

  - arguments, flags

```
ls -lt *.c
```

  - conditional
  - looping
  - ...

"EXTENSIBLE"

- add a new command with

```
cc avg.c
mv a.out avg
```

- also can add new commands with

```
chmod 755 doit
doit
```

where "doit" is a file with shell commands

Primary use

low-overhead "programming" to
manipulate files
invoke commands

Outline

- ~~Background~~
- ~~Files~~
- ~~Processes~~
- ~~Interactions~~
- **Conclusion**

Choose Your Weapons Wisely

- C or Csh? “System programming” or “scripting”?
- Abstractions -- how to make big boxes using small ones
  - System programming (makes component boxes)
    - Compiled, rich types
    - Good for creating components which demand high performance or involve complex algorithms
  - Scripting (glues component boxes together)
    - Interpreted, manipulates strings, less efficient
    - Good for gluing together existing components
    - Rapid development for gluing and GUI