Lecture P3: Unix

Overview

Background

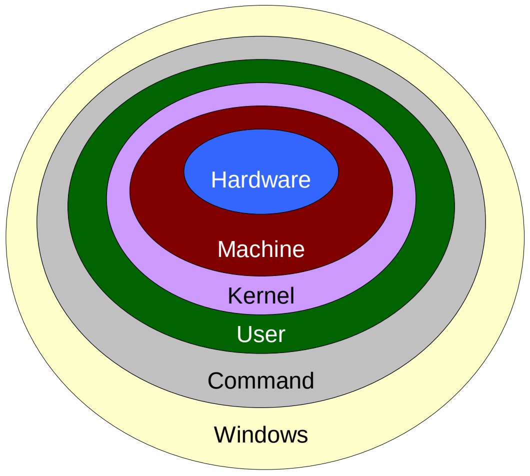
Files
- Abstraction for storage (disks).
- File manipulation commands.

Processes.
- Abstraction for processor (CPU).
- Some useful commands.

Interactions.
- Between files and processes.
- I/O redirection and piles.

Layers of Abstractions in Unix

Bare hardware.
Machine language.
Kernel.
User level (C programming)
Command level (shell).
Window system.



Operating Systems

What does an OS do?
- Makes lives easier: hides low level details of bare machine.
- Makes lives fairer: arbitrates over competing resource demands.

What we learn here.
- User level (C programming).
- Command level (shell).

Operating Systems

Multics (1965-1970).

Unix / Linux (Thompson and Ritchie 1969).

- Simplicity and elegance.
  - C language, bootstrapped implementation
  - integrated command structure
  - simplified, integrated file system
  - used by most programmers
- Continued development at AT&T (1970's) and "shepherding it out."
- Berkeley "BSD" (1978-1993): TCP/IP.
- Various flavors of commercial Unix (1980-1990).
- Linux gave it new life (1991 - present).

Operating Systems

Multics (1965-1970)

Unix / Linux (Thompson and Ritchie 1969)

DOS

Macintosh

Windows

- OS definition under litigation.

Files

File

- Sequence of bits.
- A simple and powerful abstraction for permanent storage (disks).
- Extended for things beyond disks.

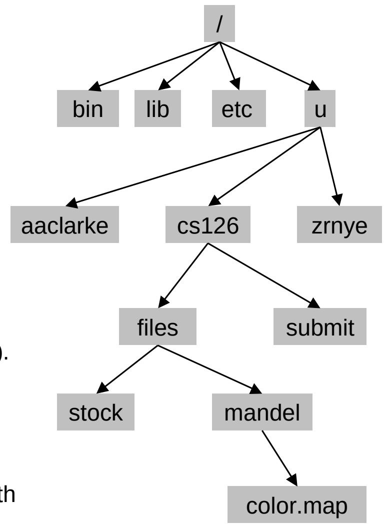
"Everything in Unix is a file."

Directory.

- Sequence of files (and other directories).
- Analog of folders in Windows / Mac.

Filename.

- Sequence of directory names on the path from "/" to the file.



File Manipulation Commands

`cat, more`	show the contents
`% more xx`	show the contents of file xx
`cp`	copy
`% cp xx yy`	copy file xx to yy
`rm`	remove
`% rm xx`	delete file xx
`% rm *`	delete all files in current directory!
`mv`	move
`% mv xx yy`	rename file xx to yy
`ls`	list file names
`% ls`	list al files in current directory
`% ls *.c`	list all files ending in .c
`% ls -tr`	list all files in reverse order of last modification

File Manipulation Commands

`mkdir, rmdir`	make or remove directory
`% mkdir hello`	make a new directory named hello
`pwd`	print name of current (working) directory
`cd`	change directory
`% cd ..`	to parent directory
`% cd ~`	to my home directory
`% cd ~xx`	to xx's home directory
`chmod`	change read/write permissions
`% chmod 600 xx`	so that only you can read/write file xx
`% chmod 700 ~`	for all files in your home directory
`% chmod 666 xx`	so that all Princeton students can access xx
`% ls -l`	list all file details (permissions, directories)

Processes

Process
- An abstraction for the processor (CPU).
- Almost every command is a process.

Over 2500 standard commands.
- Thousand more available.
- EXTENSIBLE: can even add your own.

Unix Commands

`lpr`	send file to printer
`% lpr hello.c`	print file hello.c
`man, apropos`	online documentation
`% man ls`	get help on using ls command
`cal, date, xclock`	time utilities
`% cal 2 2000`	display calendar for February, 2000
`% date`	display current date
`bc, xcalc`	calculators
`% xcalc`	graphical version of scientific calculator
`maple, matlab`	scientific computing

Unix Commands: Text Processing

`grep, awk, perl`	pattern matching
`sort`	sort the lines of a file
`diff`	print out any lines where two files differ
`emacs, latex`	text processing
`% emacs hello.c`	edit file hello.c
`ispell`	text processing
`% ispell readme`	spell-checker

Unix Commands: Programming

`emacs, xemacs`	text processing
`% emacs hello.c`	edit file hello.c
`cc, lcc, gcc,`	C compilers
`g++, java`	C++, Java compilers
`% gcc hello.c`	compile C program hello.c
`% gcc -Wall hello.c`	ask compiler to provide more warnings
`lint`	C program verifier (get even more errors)
`% lint hello.c`	check for errors in file hello.c
`gdb, jdb`	C and Java debuggers

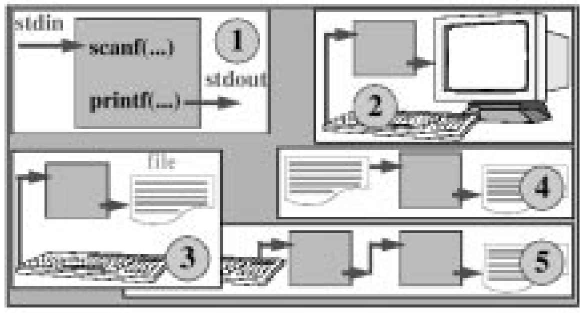
Unix Commands: Multimedia

`acroread, ghostview`	display documents
`% ghostview xx.ps`	display PostScript file xx.ps
`% acroread  yy.pdf`	display Acrobat file yy.pdf
`xv, gs`	display graphics
`% xv giraffe.gif`	display graphics file giraffe.gif
`% gs xx.ps`	display PostScript graphics xx.ps
`xfig`	create figures
`audiotool`	play or record music
`soffice`	StarOffice: free Microsoft Office clone

Unix Commands: Communication

`mail, pine`	email
`rn`	read newsgroups
`netscape`	browse web
`telnet, rlogin, ssh`	login to remote computer
`ftp`	download files

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
```

Piping:

- Connect standard output of one command to standard input of the next.

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

Don't confuse redirection and piping.

```
plotprog > lpr
```

3

4

5

Multiprocessing

Abstraction provided by operating system.

- MULTIPLE "virtual" machines for your use.
- Outgrowth of 1960s "time-sharing."
- Not found on 1st-generation PC operating systems.

Multiple windows "active"??

For COS126

- One window for editor.
- One window for UNIX commands.

Note: can use ctrl-Z and bg instead of &

Unix

```
% emacs hello.c &
  [1] 18439

% netscape &
  [2] 18434

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

Ampersand indicates "do this in the background"

Shell

Shell

- The program that's running inside your terminal window.
- Much more than just manipulating files and launching programs.
- It's an "interpreter" with its own powerful programming language.

```
#!/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 details.

Shell

Command interface to UNIX.

Just another programming language.

- sequence of instructions
- variables
- branches, loops

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

EXTENSIBLE: add another command.

- rename a.out

Unix

```
% lcc avg.c
% mv a.out avg
% gamblerall | avg | lpr
```

Shell

EXTENSIBLE: add another command.

- rename a.out
- or `chmod 700` a file containing shell commands

Shell program to annoy Steve with email

repeat 5 times → email Steve → wait 1 minute →

```
#!/bin/csh -f
@ n = 0
while ($n < 5)
  printf ”from Kevin’s class\n" |
    mail -s “yo steve!”
    stephen_w_gulyas@groton.pfizer.com
  @ n = ($n + 1)
  sleep 60
end
```

Primary use.

- low overhead “programming” to manipulate files and invoke commands

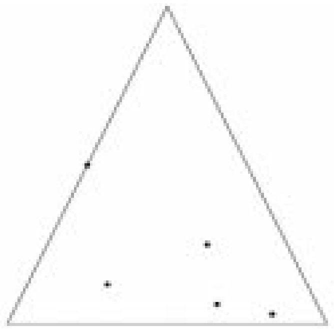
Graphics

ANSI C does not directly support graphical output.

- Need help from operating system.
- In this course we use “PostScript” to get cool pictures.
- Don’t worry about details yet.

Unix

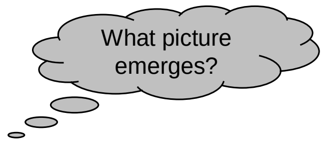
```
% cat ifs.ps

  %!
  50 200 translate
  /pt {0 360 arc fill} def
  125.0 250.0 5.0 pt
  312.5 125.0 5.0 pt
  156.2  62.5 5.0 pt
  328.1  31.2 5.0 pt
  414.1  15.6 5.0 pt
  showpage

% gs ifs.ps
```

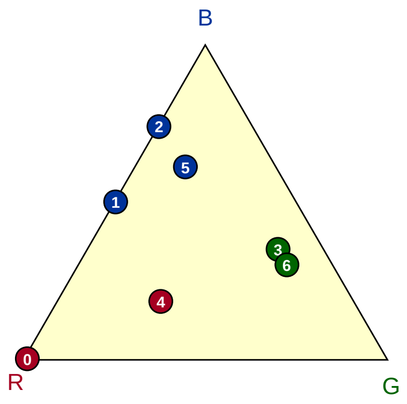


Graphics

Game played on equilateral triangle, with vertices R, G, B.

- Start at R.
- Repeat the following:
  - pick a random vertex
  - move halfway between current point and vertex
  - draw a “dot” in color of vertex





Graphics

ifs.c

```
#include <stdlib.h>
#define N 50000

int main(void) {
  int i, r;
  double x = 0.0, y = 0.0, x0, y0;

  for (i = 0; i < N; i++) {
    r = randomInteger(3);
    if (r == 0)      { x0 =   0.0; y0 =   0.0; }
    else if (r == 1) { x0 = 500.0; y0 =   0.0; }
    else             { x0 = 250.0; y0 = 500.0; }
    x = (x0 + x) / 2.0;
    y = (y0 + y) / 2.0;
    printf("%f %f\n", x, y);
  }
  return 0;
}
```

Graphics

Text output is boring.

- Add printf statements to create PostScript.
- Use gs to view PostScript file.

ifs.c

```
. . .

printf("%%!\n 50 200 translate\n"
       "/pt {0 360 arc fill} def\n");

printf("0 0 moveto 500 0 lineto"
       "250 500 lineto closepath stroke\n");

for (i = 0; i < N; i++) {
  . . .
  printf("%f %f 1.0 pt\n", x, y);
}

printf("showpage\n");
```

Conclusions

Choose your weapon wisely.

- C vs. Shell
- Systems programming vs. scripting.

Abstractions: how to make big boxes using small ones.

- Systems programming: makes component boxes.
  - compiled, rich types
  - good for creating components which demand high-performance or complicated algorithms
- Scripting: glues component boxes together.
  - less efficient since interpreted not compiled
  - good for gluing together existing components
  - rapid development for gluing and GUI