COURSE INTRODUCTION

Software Tools

EECS2031 Winter 2018

Manos Papagelis

Thanks to Karen Reid and Alan J Rosenthal
for material in these slides

What EECS2031 is about?

- A useful way to think about this course is that it is about the **environment** in which your programs run
  - understanding the environment
  - developing tools:
    - for interacting with the environment
    - for getting information about it
    - for influencing it
  - learning a new language (or two) to help us

Course Overview

- Part I (UNIX, Shell Programming) ~3 Weeks
  - UNIX
  - Understanding the Shell and Shell Programming
- Part II (C Programming) ~7 Weeks
  - C Fundamentals, Input/Output
  - Expressions, Selection Statements, Loops, Types
  - Arrays, Functions
  - Pointers, Arrays, Strings
  - Structures, Dynamic Memory Management
- Part III (UNIX Programming) ~2 Weeks
  - Processes, Signals, Pipes

Self Study Topics

- Using Unix - some tutorial coverage
- Using software tools
  - an editor – vi, emacs, nedit, …
  - a debugger – gdb, …
  - an IDE – eclipse, …
- Readings

Environment

- Environment: EECS Computing Facility
  - UNIX/LINUX system
  - SSH to eecs.yorku.ca
  - Use your EECS login and password

Windows & Mac Users

- Windows: If you want to do some of your work on your own machine, you will need to install cygwin:

http://www.cygwin.com/

- MacOS: Use the "Terminal" application

For my interest

- How many of you have UNIX/LINUX knowledge?
- How many of you have done some shell scripting?
- How many of you have programmed in C or had attended an introductory course in C?
- How many of you have understanding of processes, pipes, signals in UNIX environment?

Today's Overview

- Course Administrivia
- Unix & Unix as a File System
- The Big Picture

EECS2031 Administrivia

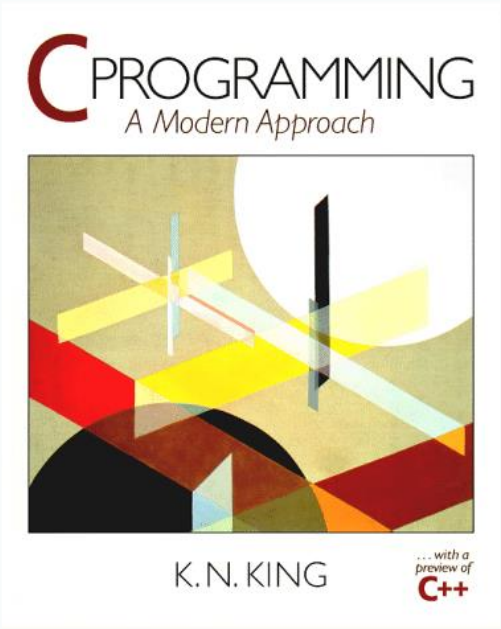
Course Information

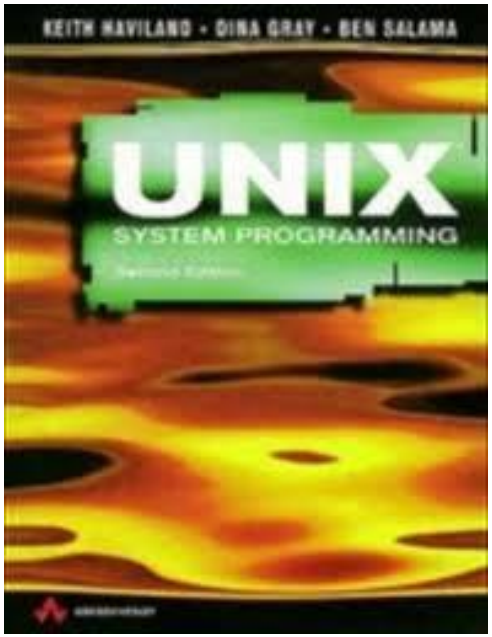
- Lectures (CLH E):
  - Tue, 9:30-10:30am
  - Thu, 9:30-10:30am
- Tutorials/labs (LAS1006):
  - Lab01: Tue, 13:00-15:00
  - Lab02: Wed, 13:00-15:00
- Course Website (soon online):
  https://www.eecs.yorku.ca/~papaggel/courses/eecs2031/

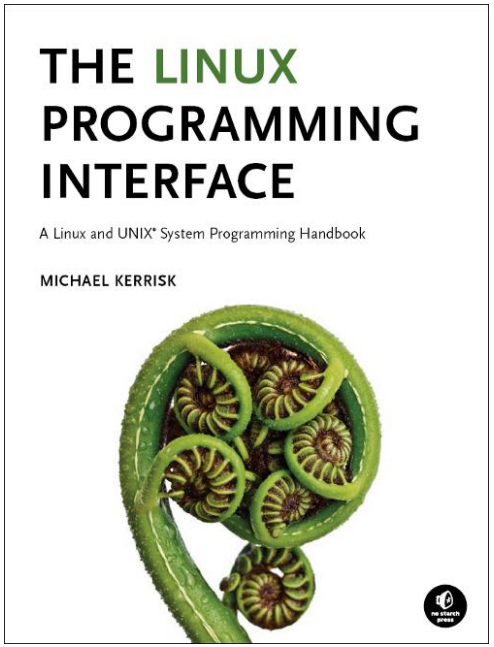
Communication

- Office hours:
  - TR 10:30am – 11:30am
  - by appointment in special cases
- Email:
  - Subject must include **EECS2031**
  - Email is a formal method of communication:
    - State your question clearly, with enough context
    - Sign it (Name, login and student # are the most useful)

Course Textbooks







- C Programming: A Modern Approach, Second Ed., K.N. King. W. W. Norton and Company, 2008.
- (optional) Unix System Programming Second Edition, Keith Haviland, Dina Gray, Ben Salama. Addison-Wesley, 1998.
- (optional) The Linux Programming Interface, Michael Kerrisk. No Starch Press, 2010.

Assignments

- A1: Shell Use and Programming
- A2: Tools in C (Loops, Arrays, Strings)
- A3: More tools in C (Dynamic Memory Management, Files, Linked Lists)

All assignments, tests and exam are **individual work**

Assignment Policies

- Assignments are due at 11:59 p.m. on the due date - check website for final due dates
- Late Assignment Policy: **3 grace days**
- Code **must** work on EECS servers
- Marking
  - assignment 1, 2, 3: (mostly) based on auto-markers
- Code that does not compile gets zero

Did you catch that?

Code that does not compile will receive a grade of 0

Submitting Assignments

- You will be using the **submit tool** to manage and submit your assignments
- Details will be provided on how to submit your assignments
- Do not wait until the last minute to try to commit your assignment for the first time

Plagiarism

- "The work you submit must be your own, done without participation by others. It is an academic offense to hand in anything written by someone else without acknowledgement."
- You are not helping your friend when you *give* them a copy of your assignment
- You are hurting your friend when you *ask* them to give you a copy of their assignment

What is Cheating?

- Cheating is
  - copying parts or all of another student's assignment
  - including code from books, web sites, other courses without attribution
  - getting someone else to do substantial parts of your assignment
  - giving someone else your solution
- Cheating is not
  - helping to find a bug in a friend's code (be careful)
  - helping each other understand man pages or example code

A few do's and don'ts

- Do
  - ask questions if you don't understand something
  - work together to understand concepts/assignments
  - use tutorials/labs and office hours
  - read textbook or provided online material before class
- Don't
  - hand in other peoples' work (it's cheating)
  - harass others (see the University's policies)
  - distract or disrupt the class (it's immature)

Course Marking Scheme

Work	Weight	Comment
3 Assignments	45%	15% each
Midterm Test	15%	In-class, paper-based
Final Exam	40%	You must get **>=40% in the final exam** to pass the course

UNIX

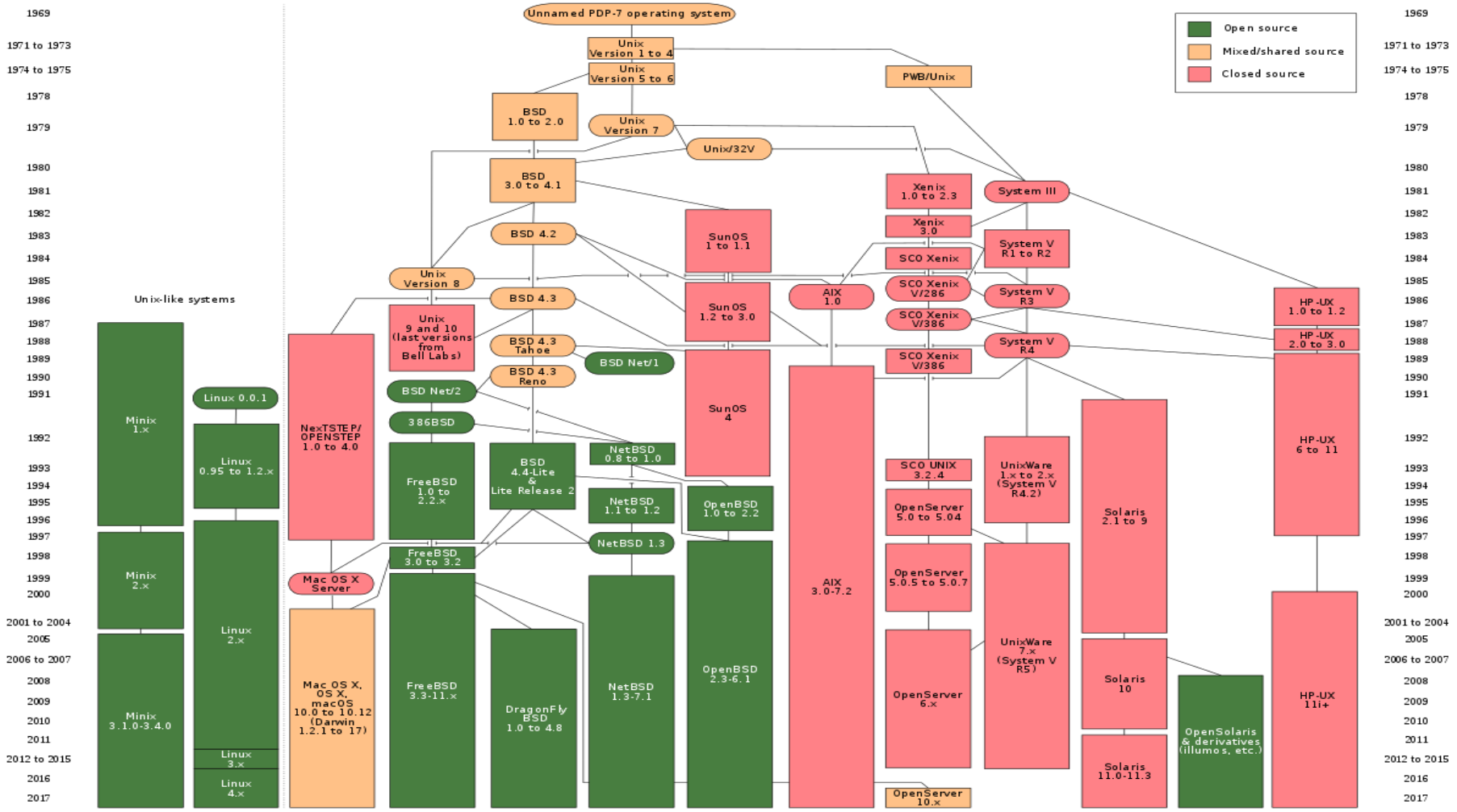
Unix History

- Developed in 1969 (in assemply) by a group of AT&T employees at Bell Labs, including Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, Michael Lesk and Joe Ossanna.
  - *Dennis Ritchie* and *Ken Thompson* ported an enhanced version to a PDP-11/20 in 1970.
  - Ritchie and Rudd Canaday ported a cut down version of the BCPL language to Unix, calling it **B**.
- Pipes and **C** (successor of **B**) were added in 1971-73
- "License to universities, but no support.", BTL Lawyers
  - This led to extensive sharing

More Unix History

- Canadian connection: Brian Kernighan, Rob Pike, Bill Reeves, ...
- Berkeley Software Distribution grew out of collecting and distributing bug fixes (Led to FreeBSD, NetBSD)
- Bill Joy started at Berkeley but joined the startup Sun Microsystems in 1982
- 1991, Linus Torvalds (Linux kernel initiator) posts a note describing his experimental OS modeled on Minix (Unix-like OS)

Evolution of Unix and Unix-like Systems



Minix Linux MacOSX FreeBSD Android …

Why Unix?

- Multi-user, multi-tasking computer operating system
- Available on a number of platforms
- Shares computer resources sensibly
- Permits manipulation of files, processes, and programs
- Allows inter-process and inter-machine communication
- Permits access to its operating features

The Unix Philosophy

- Write programs that **do one thing** and **do it well**
- Write programs to **work together**
- Write programs that handle **text streams,** because that is a universal interface

Ways of Looking at a System

- Some of the ways we look at UNIX:
  - As an end user
  - As an environment for programs to run
  - As a file system - part of the overall environment

UNIX: End-user Interaction

- Unix has a rich set of tools for dealing with its own structures and data:
  - need to be familiar with them to manage (your portion of) the system
  - you may already know some (i.e., move around filesystem, list, copy and remove files, run programs and performing other tasks)
- Involves learning how to write *UNIX shell scripts*

UNIX: Environment for Programs

- How programs get ready to run
- What happens when a program is run
- What happens when your program writes to/reads from a file
- How your code can start other pieces of code and interact with them
  - how programs "talk" to each other
  - how programs "talk" to the outside world (networks)

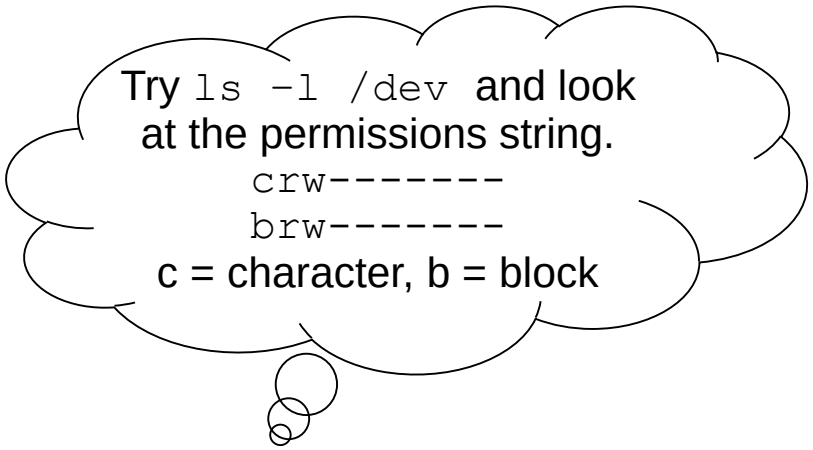
UNIX: As a File System

- What are files? what are directories?
  - how are they organized, maintained?
  - what information is accessible about them?
- What different file types are there?
- How to access them?

Unix as a File System

Files and Directories

- "Everything is a file."
- Unix provides a **file interface** for all Input/Output.
  - regular files
  - directories
  - devices
    - video (block)
    - keyboard (character)
    - sound (audio)
    - network (block)

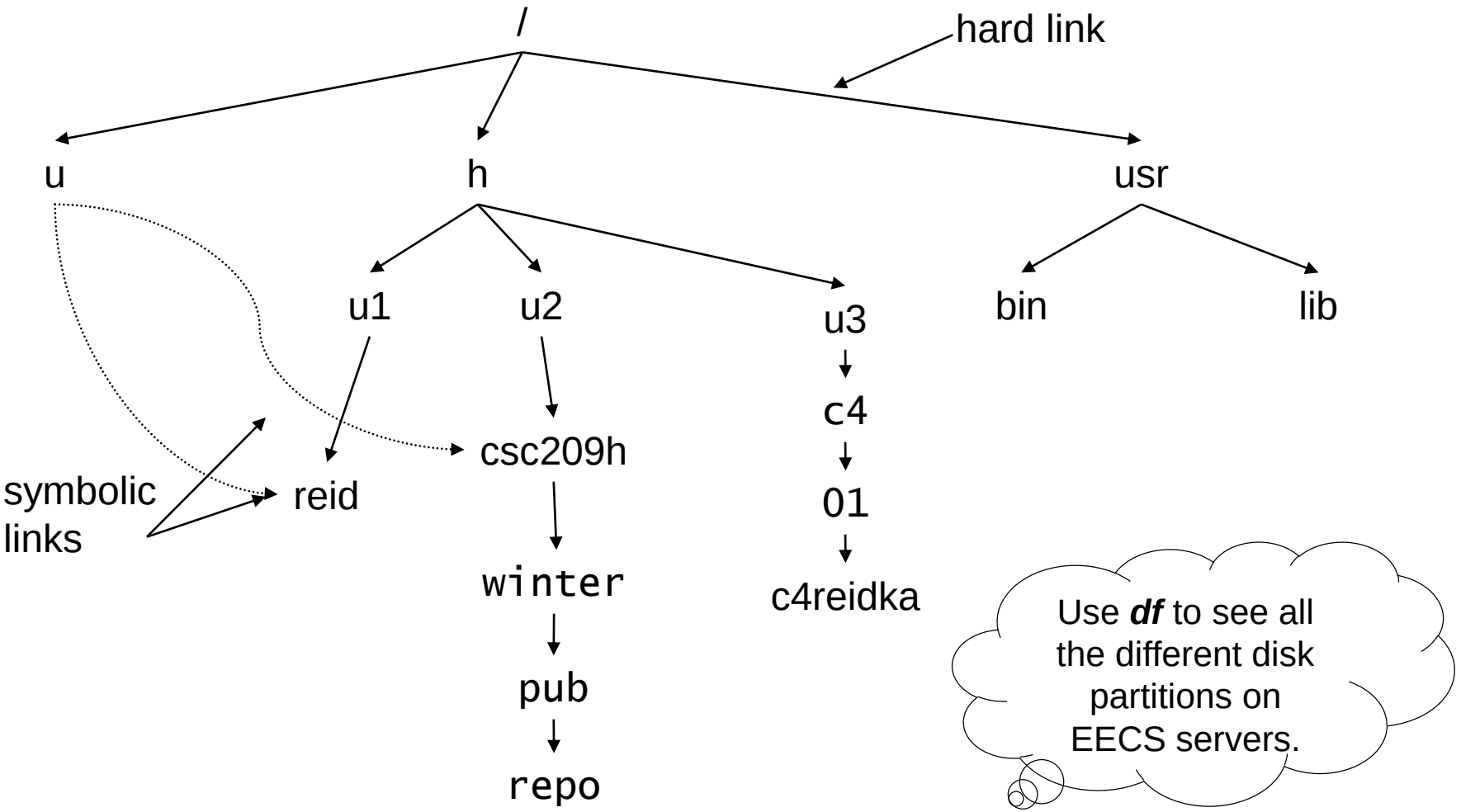


- File interface = open, read, write, close

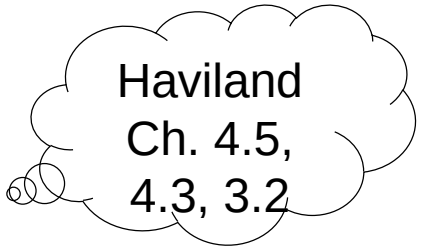
File System Hierarchy

- Everything starts in the "root" directory whose name is "/"
- A directory is a file that contains directory entries
- A directory entry maps a file name to an inode
- An inode is the data structure that contains information about a file, including which disk blocks contain the file data

File System Hierarchy



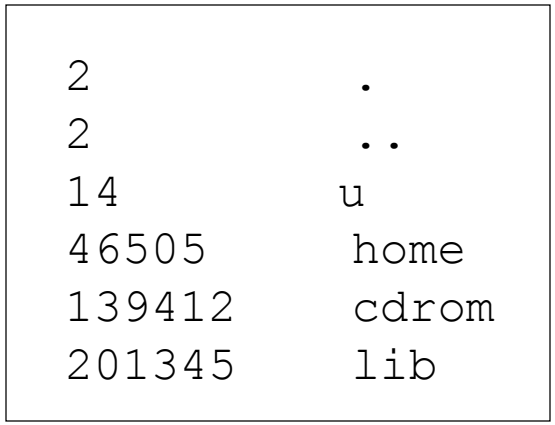
File Systems and Links



- One file system per disk partition
- A file system can be mounted at any point in the directory tree of another file system
- A hard link is an entry in a directory file which specifies an inode
  - There can be several hard links to a file, but hard links cannot cross file systems
- A soft link (symbolic link) is a small file containing the path name of the linked file or directory
  - Soft links work across file systems

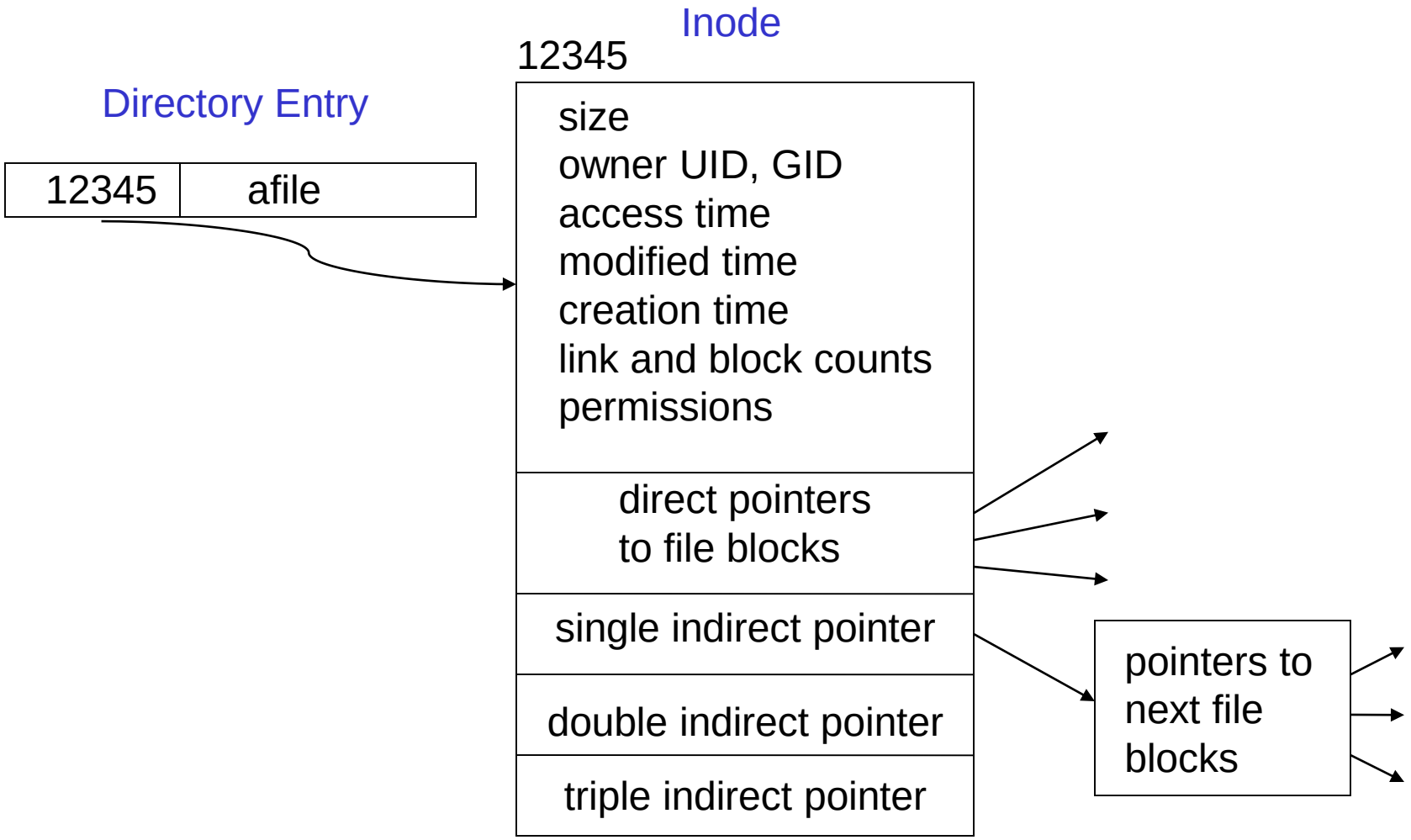
Directories and Links

directory file

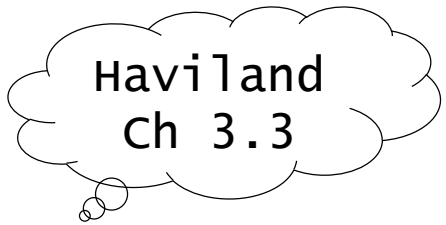



```
% ls -l /
drwxr-xr-x    2 root   root   4096 Nov  8 17:56 bin/
drwxr-xr-x    2 root   root   4096 Aug 10 14:46 cdrom/
drwxrwsr-x    2 root   staff  4096 Feb  8  2002 home/
drwxr-xr-x    6 root   root   4096 Sep  2 15:26 lib/
lrwx------    1 root   root      6 Sep  2 15:32 u -> /cdf/u/
```

Inodes and Directory Entries



Stat

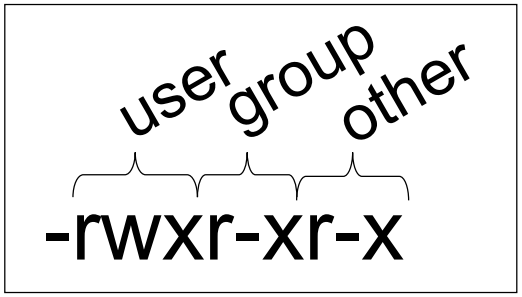


stat(): A Unix *system call* that returns useful data about a file inode

```
greywolf% stat csc209h
 File: `csc209h'
  Size: 512               Blocks: 2          IO
  Block: 8192   directory
Device: 16h/22d Inode: 27612       Links: 7
Access: (0755/drwxr-xr-x)  Uid: (    0/
  root)   Gid: (  517/ csc209h)
Access: 2010-01-06 11:32:44.293409000 -0500
Modify: 2010-01-04 12:06:15.987312000 -0500
Change: 2010-01-04 12:06:15.987312000 -0500
```

Permissions


```
-rwxr-xr-x 1 reid   0 Jan  6 11:35 allexec*
-r--r--r-- 1 reid   0 Jan  6 11:35 allread
dr-xr-xr-x 2 reid 512 Jan  6 11:36 dir-read/
dr-x--x--x 2 reid 512 Jan  6 11:36 dir-search/
-rw------- 1 reid   0 Jan  6 11:35 ownerread
-r--r--r-- 1 reid   0 Jan  6 11:35 readonly
```



- File permissions
  - read, write, execute – pretty much what you think
- Directory permissions
  - read: you can "read" the file (run ls, cat, etc.)
  - write: you can "write" (create/edit/delete) the file
  - execute: you can "execute" the file
- Use *chmod* to change file permissions
  - e.g.: % chmod 664 myfile

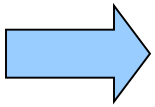
THE BIG PICTURE

The Big Picture

Source code file

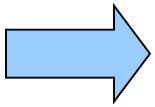
hw.c

```
#include <stdio.h>
void
main()
{
   printf("Hello world");
}
```



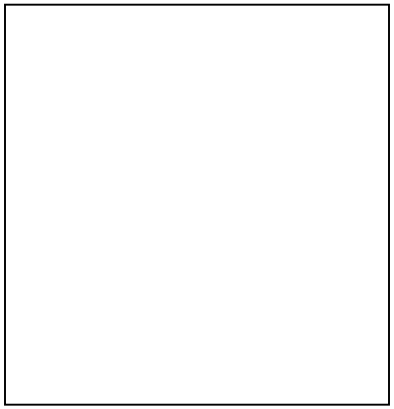
Compile it

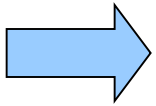
% gcc –o hw hw.c



Object file

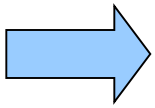
hw



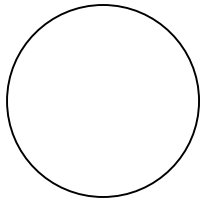


Run it

% hw



Process



Source Code Files

hw.c

```
#include <stdio.h>
void
main()
{
    printf("Hello world");
}
```

- What is a file?
  - Sequence of bytes
- A file system?
  - A hierarchy of files + tools
- How does the system know where to find `hw.c`?
  - paths, working directories, …
- What is the meaning of `#include<stdio.h>` ?
- What does `printf` really do?

Compiling a program

```
% gcc –o hw hw.c
```

- A compiler is a program that translates source code into object (machine) code
- Here we are running the compiler at the command line

The Shell

```
% gcc –o hw hw.c
```

- The % is a shell prompt
- The shell is a program that can execute another program
- The shell
  - accepts commands (programs) as input
  - finds the executable
  - interprets the arguments
  - starts executing the command
- The shell also has some “built-in” commands

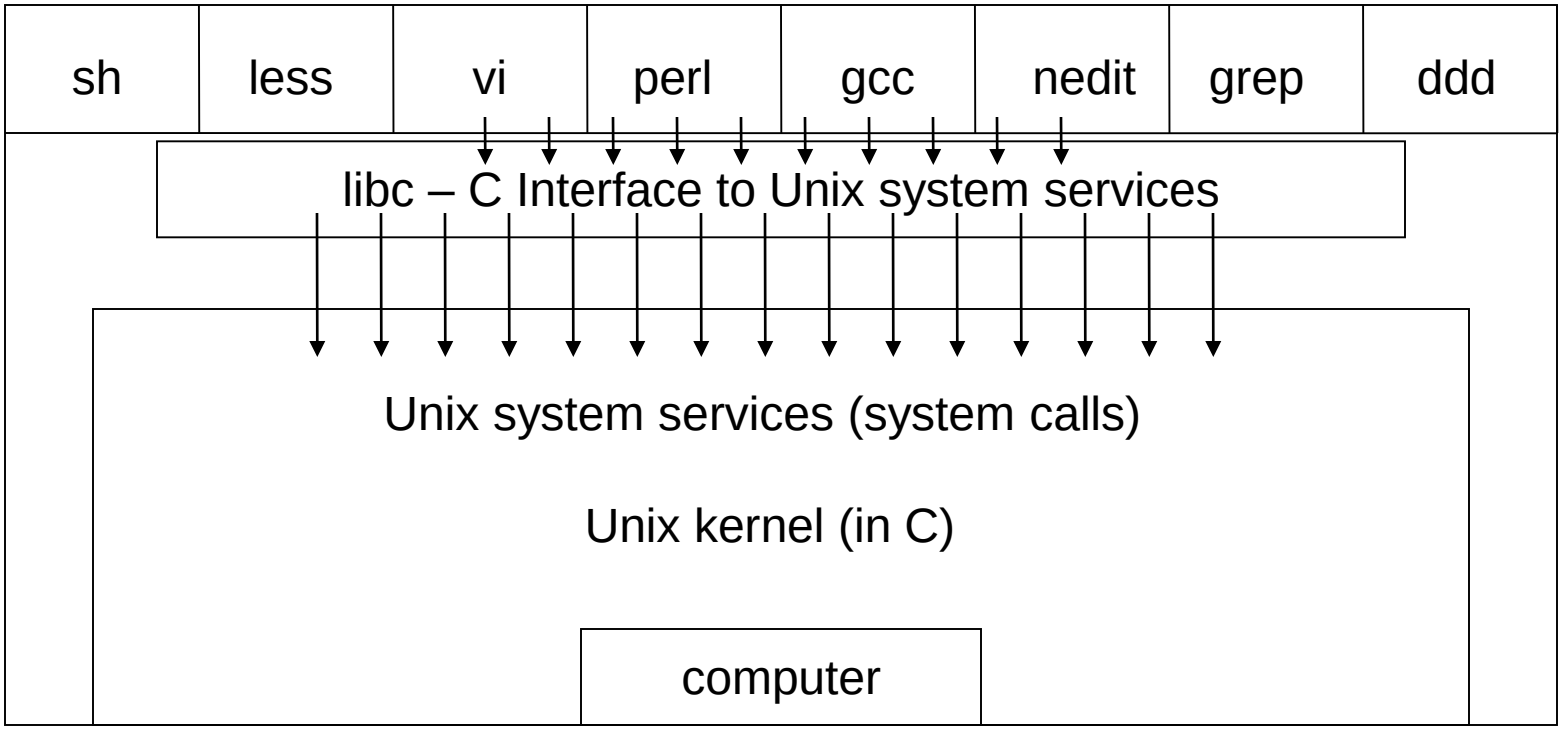
Running a program

```
% gcc –o hw hw.c
```

```
% hw
```

- After we have compiled the program, we can run it
- load a program into memory and hand it off to the OS that takes control of running it

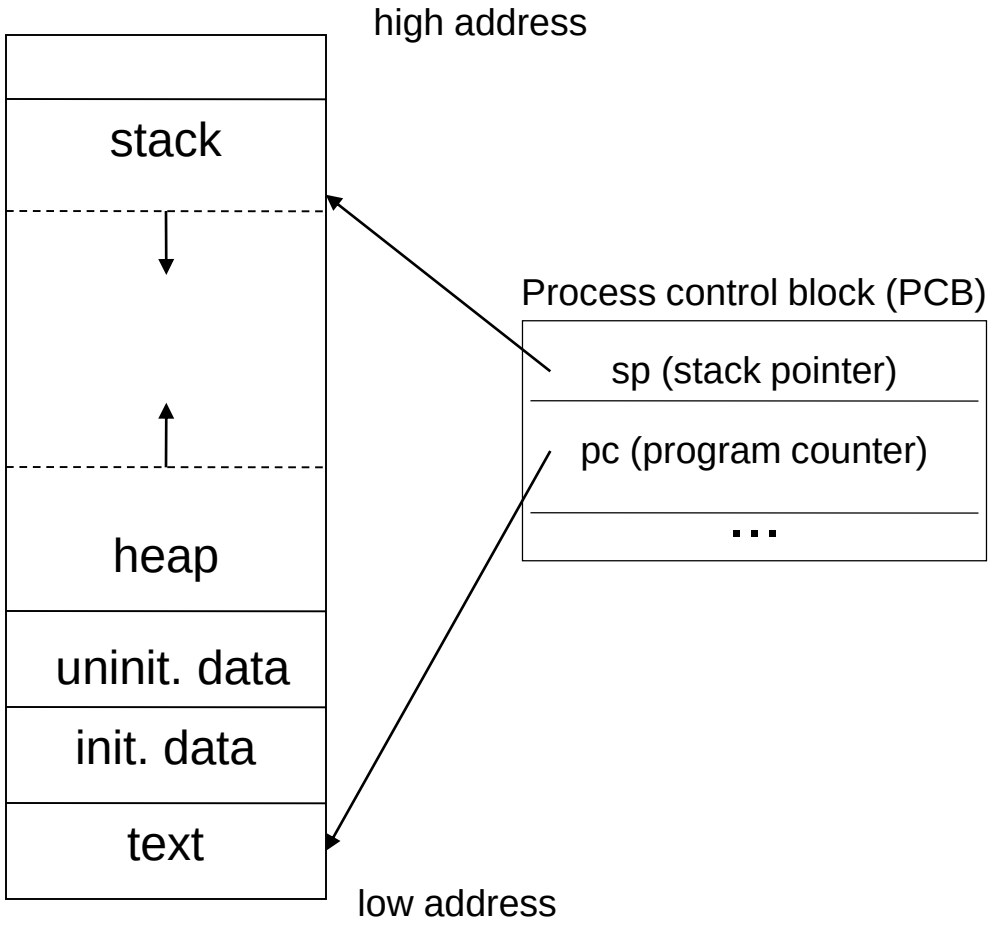
A Different Big Picture



Processes

- A **process** is an executing instance of a program
- The OS keeps track of information about the process
  - process ID – a unique non-negative integer
  - process state – "running", "ready", "blocked"
  - program counter – which instruction is being executed
  - a list of open files
  - etc.

Object Files/Executables



- Typical memory layout of programs.
- The kernel keeps a PCB for each process

What is Next?

- Shell & Shell Programming
- Tutorial about UNIX (next week)
- Tutorial about Shell scripting (week after)