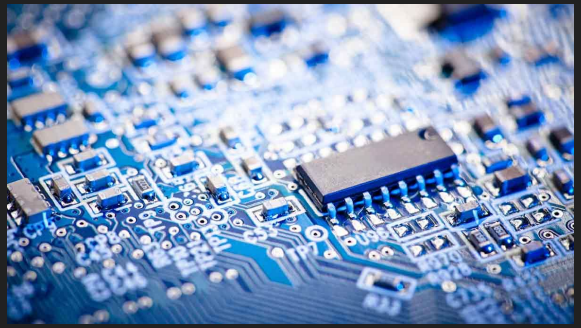
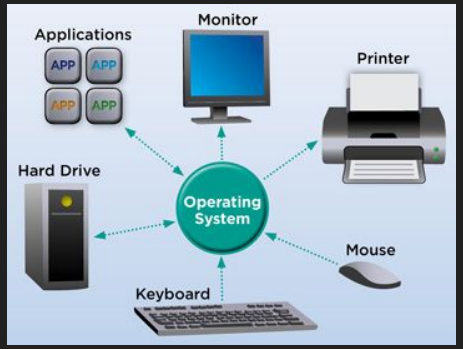
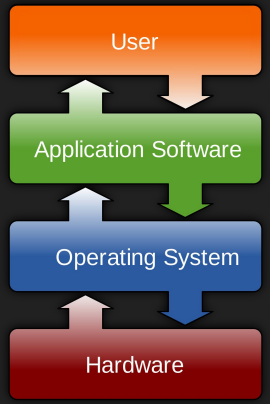
Please do not redistribute these slides without prior written permission





CS3650 Computer Systems

Dr. Alden Jackson



Lecture 1 - Logistics & Overview

Professor Alden Jackson, PhD

About your Instructor

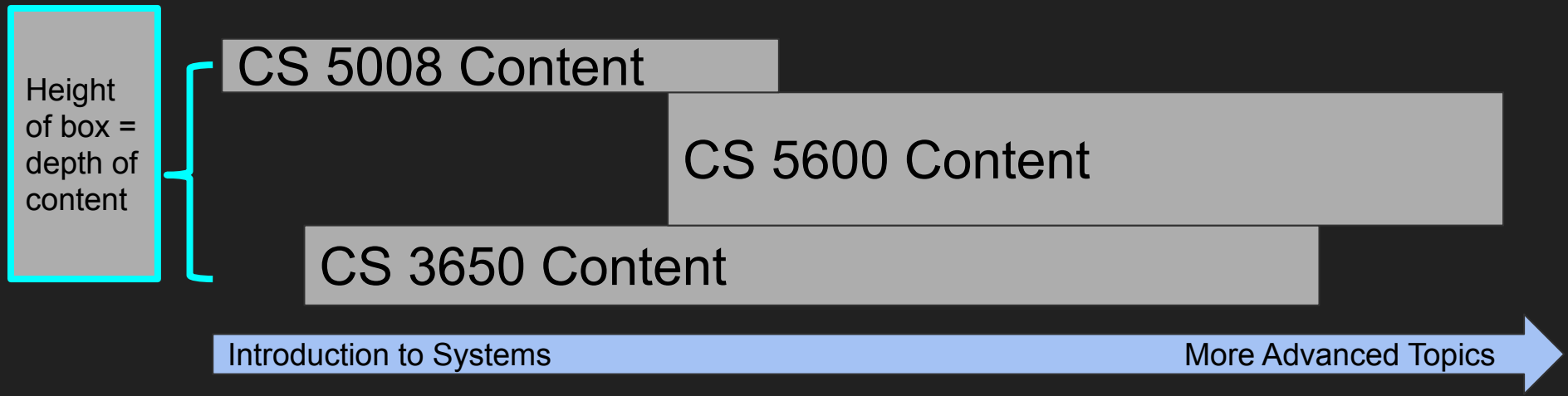
- I grew up in Maryland, just outside Washington, DC.
- BA in Physics from the University of Dallas.
- I enjoyed modeling and simulating physical systems ⇒ MS in EE at Howard University.
  - Researching image processing and HW support for graphics libraries
  - Ran a research computing center
- PhD at the University of Delaware, where I found networking
- Primarily interested in ultra high speed network and router architecture, transport protocols, reliability of massive distributed systems, network security and privacy, and Internet censorship mitigation (refraction networking)

Places where I worked on systems...

So what is this course?

Computer Systems course in Computer Science

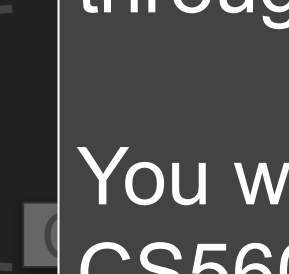
- A rough visualization of where the course is in the curriculum



Masters level course in Computer Science

- A roug

Height of box = depth of content



In

Advanced Topics

My goal is to get everyone through & not be intimidated!

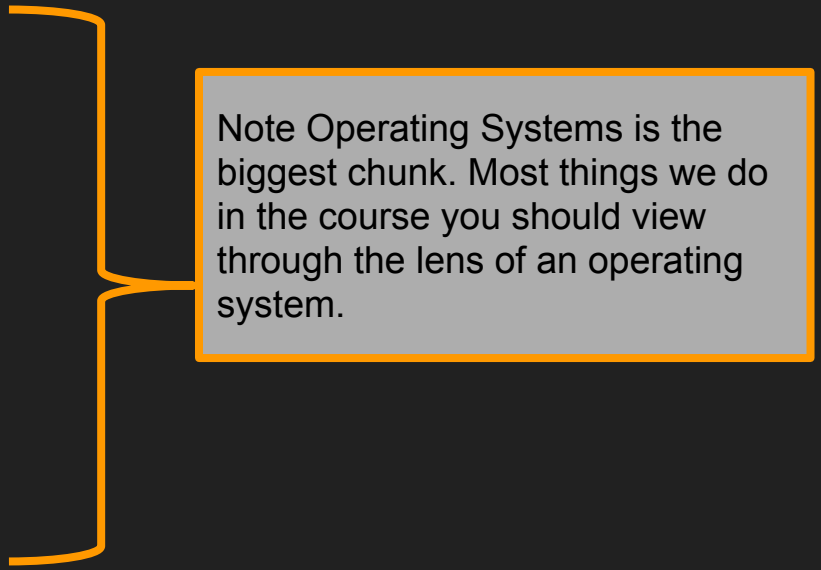
You will then be ready to take on CS5600!

Roughly Speaking this course has a few ‘modules’

1. Computer Systems Fundamentals
   a. Terminal, C, Assembly, Compilers, Tools
2. Virtualization (**is my code the only thing running on login?**)
   a. The Process
3. Computer Architecture (**Locality, Speed, Memory Abstractions**)
   a. Memory/Cache/Virtual Memory
4. Concurrency (**managing shared resources**)
   a. Threads/Locks/Semaphores
   b. Parallelism
5. Persistence (**where do my files go when the power is turned off?**)
   a. File Systems
   b. Storage Devices
6. Other Selected Topics Throughout The Semester
   a. Debugging, Instrumentation, Safety

Roughly Speaking this course has a few ‘modules’

1. Computer Systems Fundamentals
   a. Terminal, C, Assembly, and Compilers
2. Virtualization
   a. Processes
3. Computer Architecture
   a. Memory/Cache/etc
4. Concurrency
   a. Threads/Locks/Semaphores
   b. Parallelism
5. Persistence
   a. File Systems
   b. Storage Devices
6. Other Selected Topics
   a. Debugging/Instrumentation/Final



Note Operating Systems is the biggest chunk. Most things we do in the course you should view through the lens of an operating system.

Why use the operating system as the lens to learn systems?

OS as Middleware

- It abstracts low-level details and provides high-level interfaces
- Middleware is crucial because it allows programmers to build powerful software without reinventing low-level mechanisms.

Bridging Hardware and Applications

- The OS connects the physical machine to user programs, transforming raw hardware into usable resources.
- It manages CPUs, memory, storage, and networks, ensuring that applications can run safely and efficiently.

Integration of Knowledge. Studying OS ties together all layers of computer science

- Hardware/architecture: instruction sets, memory hierarchies, I/O devices.
- Systems programming: C, assembly, and low-level debugging.
- Algorithms & data structures: scheduling, synchronization, paging, file systems.

Computer Systems = Magic?

- I hate to break it to you, but there is no magic in computers.
- Computers are just 1’s and 0’s. In this course, we are going to look at 1’s and 0’s, and how to combine them to create different **abstractions**.
- That is where the magic comes in–through the creativity and the art of computer science.
- Computer Science is an art!

Course Goals

- We will review the syllabus (which on Canvas) in detail @ our next class
- Reading the syllabus is part of your 1st assignment

- https://khoury-cs3650.github.io/syllabus.html

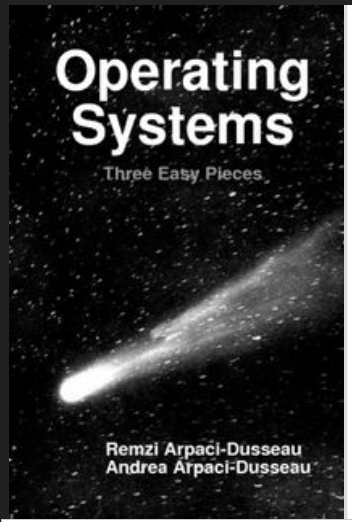
Course Materials

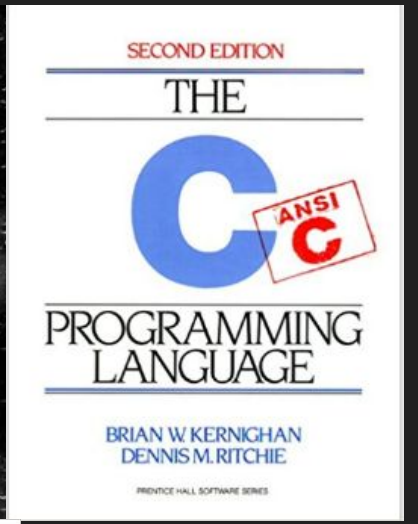
- Bringing your laptop is highly recommended (especially on lab days)
- I do not care what operating system you use on your computer
  - **You will need access the Internet and a terminal emulation program for SSH**
  - Mac (even with Apple Silicon) , Linux (Ubuntu, Debian, etc.), Windows
    - Chromebook and Surface machines are less useful
  - In the case that you do not have a laptop, Khoury has/had VDI systems that are available
    - Reach out to me about labs, where we going to try to work together in class in parallel
- However, we will use a Linux system for much of the course

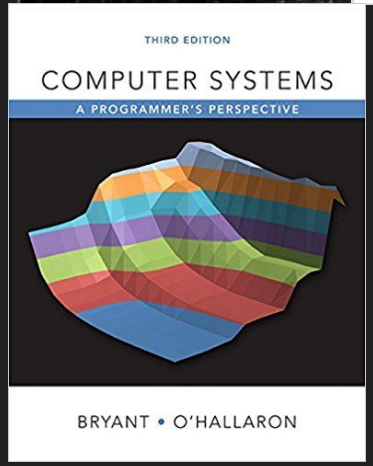


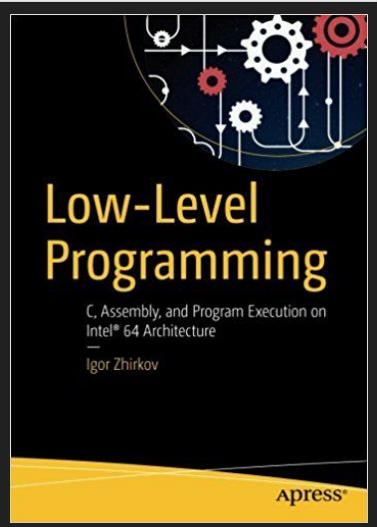
Course Text

- Operating Systems: Three Easy Pieces (aka OSTEP)
  Dive into Systems (both free)
- Low-Level Programming: C, Assembly, and Program Execution on Intel® 64 Architecture
- (Recommended texts)
  - C Programming Language Book
  - Computer Systems: A Programmer's Perspective
- Inspiration drawn from both of these texts.
- Labs and lectures will also have web resources to check out!









Teaching Style

- Everyone learns differently--optimize as needed along the way
  - There will be lectures (for auditory learners)
  - Many visuals on slides (for visual learners)
  - Labs (for kinesthetic learners)
- This is a very hands on class, we will build things–lots of things
- There will be plenty of opportunity to make mistakes

  Do not be afraid to be wrong
  - The worst case scenario is we review
- Do ask questions!
  - Occasionally I may even pause to write down the question
  - I try to avoid randomly calling on students--but do participate!
- Come to office hours! Mine or the TAs or both!

Fair Warning

- Many “PowerPoint rules” will be harmed during this course
  - i.e. There will be more than “7 words on a line” of text.
- Reason: we want the slides to be helpful for offline reading
  - (I personally get confused when I read others slides online that are sparse or have no notes)
- Slides will be generally available after class

Teaching Assistants

- Welcome them!
  - Dhairya Nirav Bhatt, Matthew Friedman, Manoj Harridoss, Manan Karnik, Ha Yoon Kim, Vishal Kumar, Collin McKinley, Preksha Sunil Morbia, Santrupti Patil, Riley Post, Lukas Tegge, and Christian Wu

- TA Office Hours: tbd
  - In Person
  - Plan is to have at least 2 TAs available 5-7PM M-F (assuming the Registrar OKs my room requests)
  - Start the week of Jan 12 (assuming the Registrar OKs my room requests)
  - Until then please post your questions on Piazza

E-mail: avoid it!

- Post general questions on Piazza
  - See Canvas for the Piazza link
  - Limit the use of private posts for general questions
  - Use cut and paste vs screen shots
  - An active Piazza is a good sign in my mind!
- Come to office hours (the prof's or the TA's) to minimize e-mail

How to ask questions

Ask specific questions

- My code doesn’t work/compile (bad)
- I tried to do A and A doesn’t work in the following ways B (error msg), C (debug info), D (certain behavior), etc. (good)
- To solve this issue with A, I tried E, F, G but did not work (good)
- Do not reveal solutions

Expectations

- You have taken some ‘programming’ related class.
  - Today you will notice I am calibrating a bit! :)
  - In the instance that you have not--you can still perform well.
    - I expect you to do the readings BEFORE class

- You know at least one programming language well
  - In this course we will use C and get exposed to x86-64 assembly
  - C is (still) the industry standard
  - (You can pick up whatever other fancy syster

Yes I know there is GO, Erlang, Rust, etc.

Evaluation

This is a course about **reading, designing and writing code**. Therefore, most of your evaluation will be on the **quality** of the code you produce and its **correctness**.

The grading distribution used for this course is below.

- 10% exercises & quizzes (approx. 10)
- 60% homework assignments
  - 40% assignments (approx. 8)
  - 20% projects (2)
- 30% exams (2)

Assignments and Projects

There will be approximately 8 programming assignments and two project assignments throughout the semester.

The first 4 assignments are to be completed individually (“solo assignments”).

The remaining assignments and projects can be optionally completed in groups. You will be responsible for selecting a partner to complete the assignments.

Projects

There will be two “projects”

These are longer (2 weeks), more **substantial** programming exercises that will **require** you to **plan** and/or **experiment** more

As such, the description will be more vague than with assignments – you are expected to do more reading, thinking, and **asking**

A note on teams

- You get to pick your partner
- You can partner up across sections
- You don’t have to keep the same partner for every project

Labs

We provide “labs” as a means to practice implementation techniques and tools

The idea is to provide exercises related to the week’s topic, which will prepare you better for tackling that or next week’s assignment/project

These will be graded mostly on effort – the intention is to encourage you to do the exercises as preparation for assignments

Ideally, we would like to provide you some class time (30-60 minutes) every week to work on the labs, but if we need more time to cover topics, the “lab” will be purely a take-home exercise

Quizzes

Almost weekly, there will be a quiz on the topics from class

The intention is to make you engage with the material

Questions will be from **lectures and readings**

Academic Integrity

Read the NEU academic integrity policies!

Here are examples for your consideration

- you work on your laptop at a library with friends and step away from your computer without locking it
- you look at your neighbors' screen/papers during an exam, but don't copy their answers
- you take a piece of code from some website and give a link to the website at the end of the homework
- you work on a homework problem with friends, type the solution at home, but it's exactly the same as that of your friend

Academic Integrity

Discussion is encouraged. But, you cannot share your code to your classmates or post them online on ANY forum.

● The university, college, and department policies against academic dishonesty will be strictly enforced. To understand your responsibilities as a student read: the Student Code of Conduct.

● Plagiarism or any form of cheating in homework, assignments, labs, or exams is subject to serious academic penalty.

● Any violation of the academic integrity policy will result in a **0** on the homework, lab or assignment, and even an **F** on the final grade. And, the violation will be reported to the Dean’s office.

How to be successful in CS 3650

Read the assigned reading **before** class

Attend the class

- Ask questions
- Answer questions

You will need the theoretical background from class to succeed in labs/assignments/projects

How to be successful in CS 3650

Labs/Assignments/Projects

- Plan ahead and start early
- **DO NOT START AT THE LAST MOMENT**
- Ask questions early

- Setting up the environment itself could take a long time
- Coding always takes longer than your expect (No one is good at this!)
- Debugging could take forever

Course Questions, Comments, Concerns?

So what exactly is C?

Here is what ‘C’ looks like

```c
#include <stdio.h>

int main(){

    puts("Hello Computer Systems!");

    return 0;
}
```

Here is what ‘C’ looks like

compile with: `gcc hello.c -o hello`

```c
#include <stdio.h>

int main(){

    puts("Hello Computer Systems!");

    return 0;
}
```

Here is what ‘C’ looks like

compile with: `gcc hello.c -o hello`

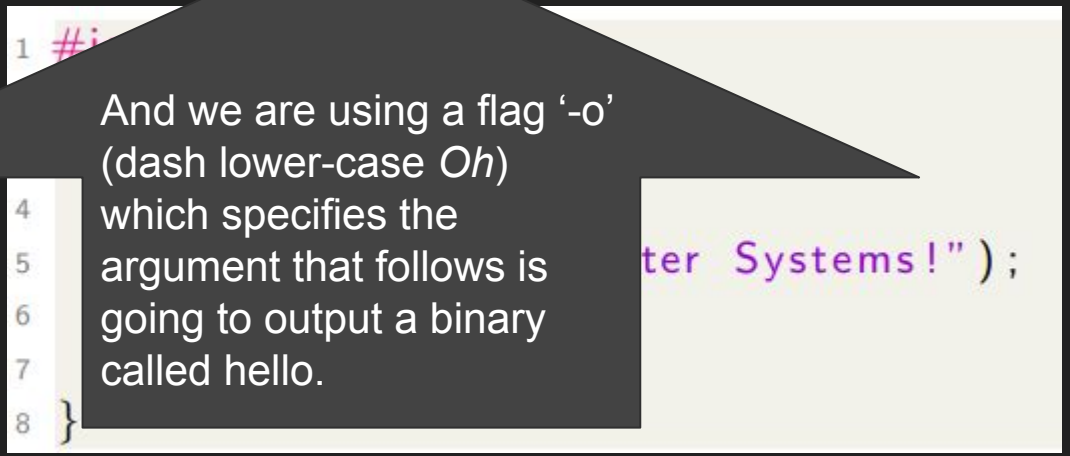
gcc is the compiler

hello.c is the name of our text source code file

```
de <stdio.h>

("Hello Computer Systems!");

rn 0;
}
```

Here is what ‘C’ looks like

compile with: `gcc hello.c -o hello`



Here is what ‘C’ looks like

compile with: `gcc hello.c -o hello`

```
1 #include <stdio.h>
2
3 int main(){
4
5     puts("Hello Computer Syst  s!");
6
7     return 0;
8 }
```

#include brings in a library of commands related to standard input and output (so we can print text to the screen)

Here is what ‘C’ looks like

compile with: `gcc hello.c -o hello`

```
1 #include <stdio.h>
2
3 int main(){
4
5     puts("Hello Compu
6
7     return 0;
8 }
```

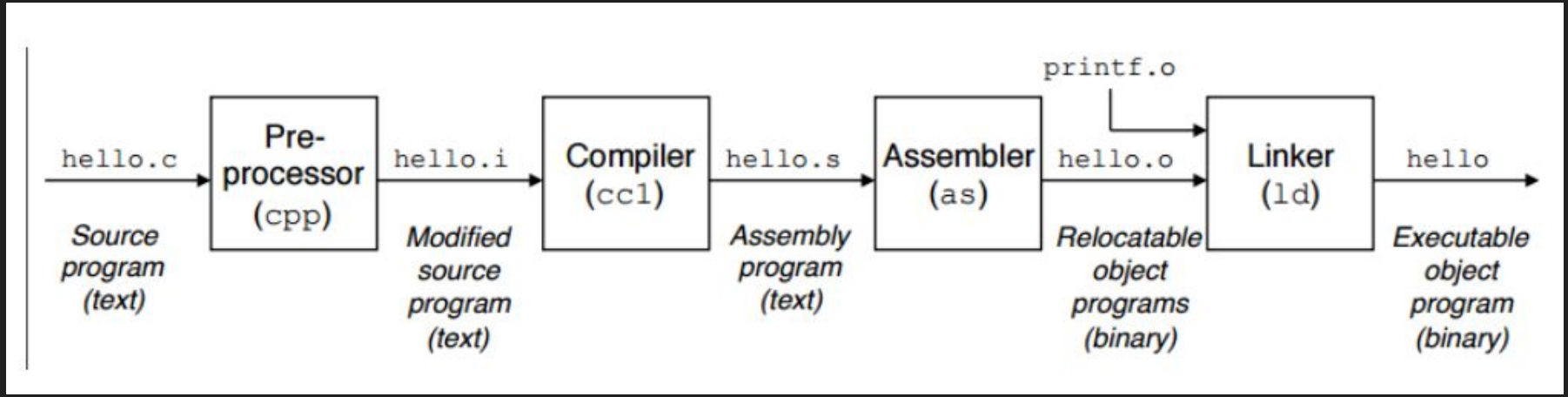
#puts prints something to the screen. *printf* will be another popular way to do this.

Here is what ‘C’ looks like

compile with: `gcc hello.c -o hello`

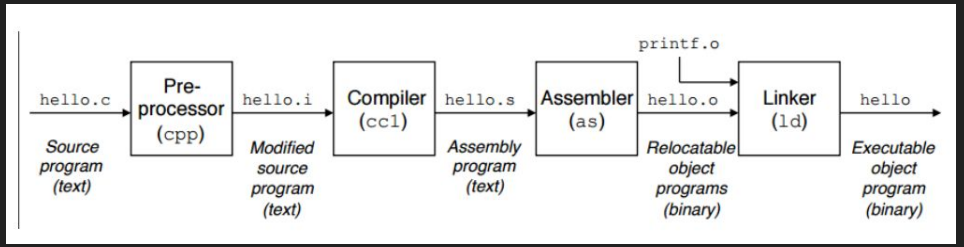
C and the compilation process

- In a picture, this is the compilation process from start to finish
- (Note in this class I'll often use clang too)



Little exercise to see what compiler is doing

- Generate assembly code
  - **`gcc -S hello.c`**
- Investigate assembly
- Compile assembly to executable file
  - **`gcc hello.s -o hello`**
- Generate Object file
  - **`gcc -c hello.s`**
- View Object File
  - **`nl hello.o (unreadable)`**
- Investigate Object File
  - **`objdump -d hello.o (disassembly)`**
  - **`objdump -t hello.o (symbol table)`**



Quick view of the assembly

- How many folks have not written assembly before?
  Close your eyes and Raise your hands

```
        .file   "hello.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello Computer Systems"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $.LC0, %edi
        call    puts
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 8.5.0 20210514 (Red Hat 8.5.0-22)"
        .section        .note.GNU-stack,"",@progbits
```

Quick view of th

- How many folks have not written assembly before?

Our string

It’s not too bad, you can pull out various functions to orient yourself

```
        .file   "hello.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello Computer Systems"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $.LC0, %edi
        call    puts
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 8.5.0 20210514 (Red Hat 8.5.0-22)"
        .section        .note.GNU-stack,"",@progbits
```

Quick view of objdump

- How many folks have not used objdump before?
  Close eyes and Raise hands again...

```
awjacks@login-students:~> objdump -t hello.o

hello.o:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 hello.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .rodata        0000000000000000 .rodata
0000000000000000 l    d  .note.GNU-stack        0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame      0000000000000000 .eh_frame
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 g     F .text  0000000000000015 main
0000000000000000         *UND*  0000000000000000 puts
```

Quick view of objdump

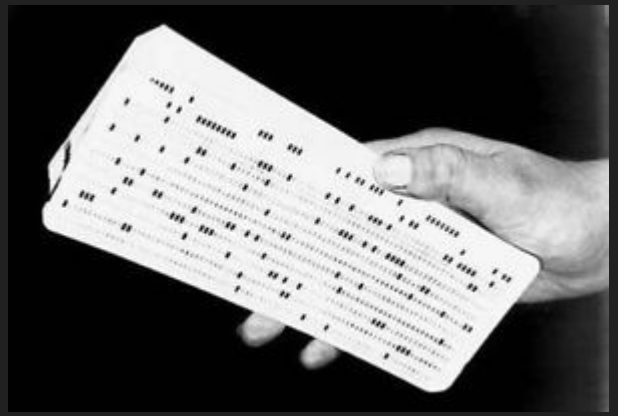
- How many folks have not used objdump before?

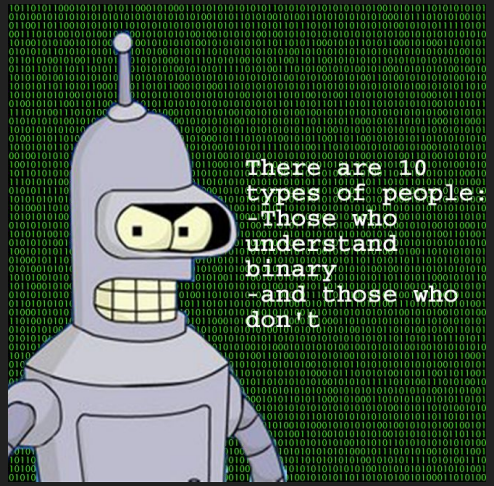
```
awjacks@login-students:~> objdump -t hello.o

hello.o:     file format elf64-x86-64

SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 hello.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l    d  .rodata        0000000000000000 .rodata
0000000000000000 l    d  .note.GNU-stack        0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame      0000000000000000 .eh_frame
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000000000 g     F .text  0000000000000015 main
0000000000000000         *UND*  0000000000000000 puts
```

Powerful tool to pull out some information
(Can see functions/libraries used)

So Compilers are pretty neat

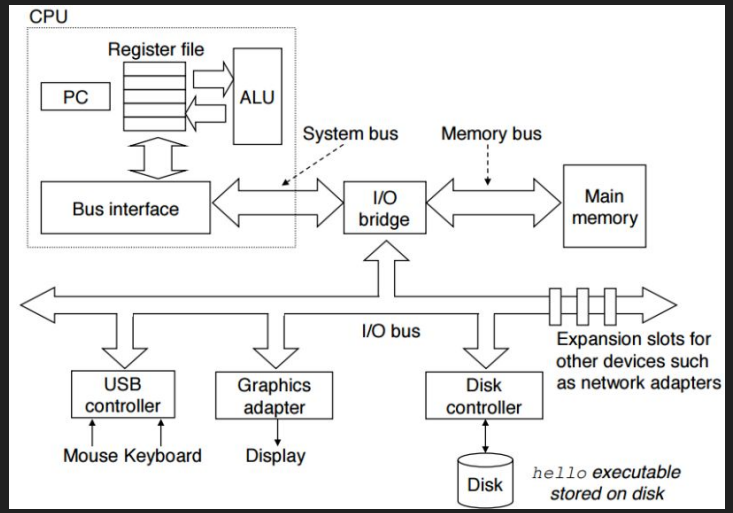
- When we start looking at some of the information taken in, we appreciate the job they do.
  - i.e. transform high level language to binary
- All of a sudden, writing some C code is not so bad!
  - (And it of course is better than pure binary!)

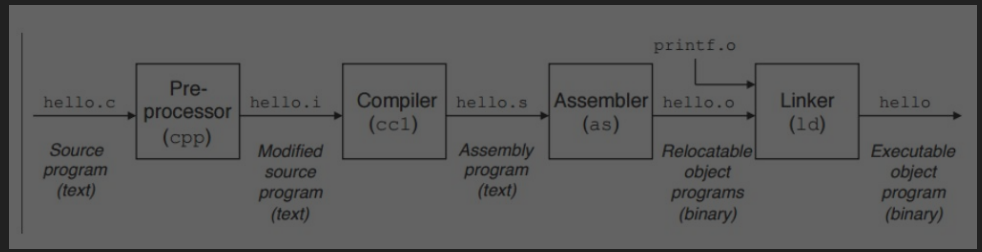




So compilers are a core element of this class

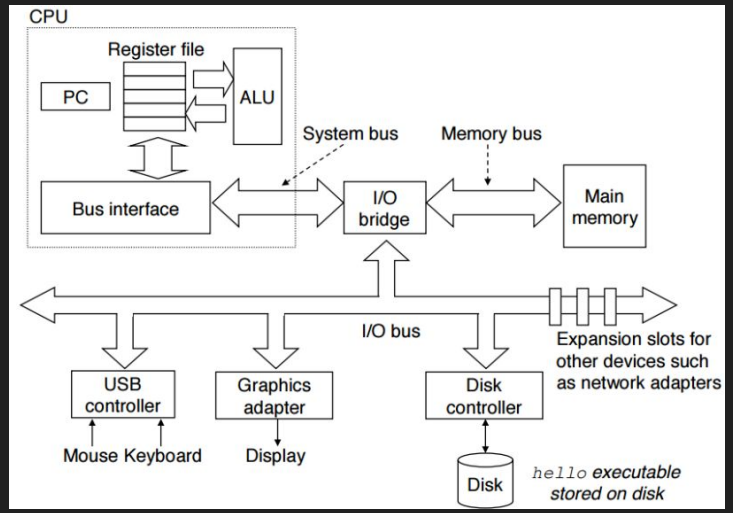
- The other core pieces are the hardware(left) and operating system (right)





So compilers are a core of this class

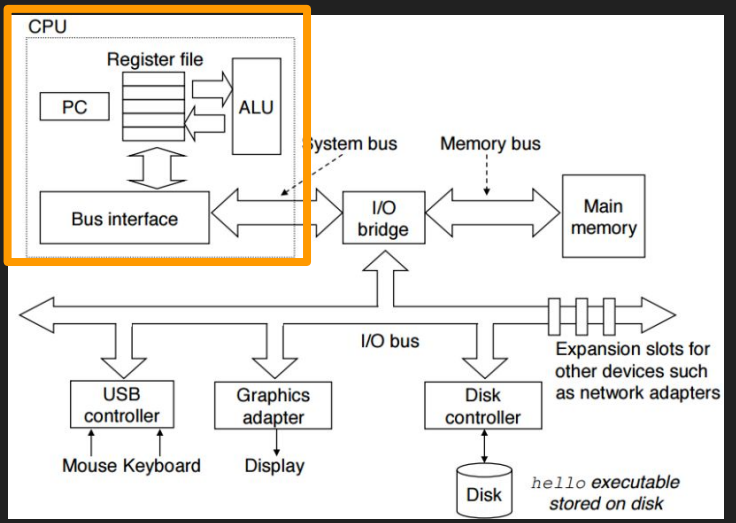
- The other core pieces are the hardware(left) and operating system (right)



Let’s take a few minutes to think about the hardware

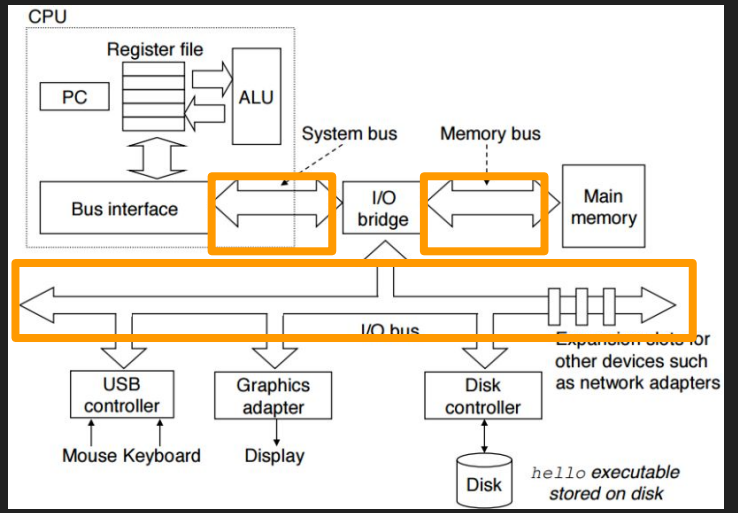
Modern Hardware Visual Abstraction

- The “brain” of modern hardware is the CPU
  - That’s where 1 instruction is executed at a time
  - Only 1!
  - (Note: Modern computers have multiple cores)
- We generally measure the speed at which a CPU executes in Megahertz or Gigahertz
  - This is a metric for how ‘fast’ a CPU performs, and how complex of software can be run.



Modern Hardware Visual Abstraction

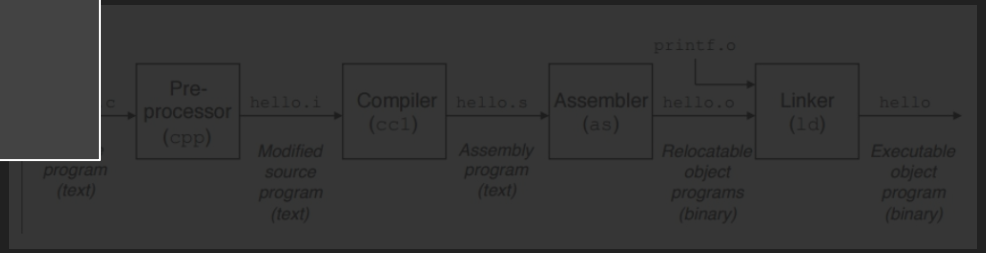
- Beyond the CPU, a number of devices may also be connected.
- Buses transfer information from devices and memory into the CPU.
- There is a lot going on, and this needs to be managed
- *Note: Busses can be thought of as simple networks, with many things hardcoded*



So compilers are a core of this class

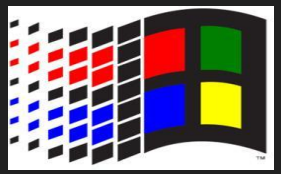
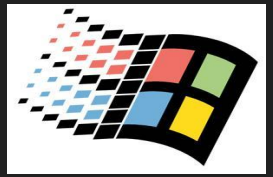
- The other core pieces are the hardware(left) and operating system (right)

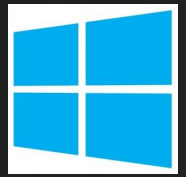
Let’s take a moment to think about operating systems



Many Different OSes

Windows

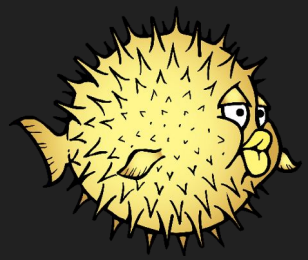


Linux





BSD

Many Different OSes

Windows

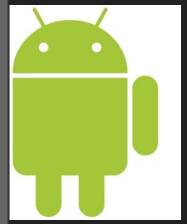




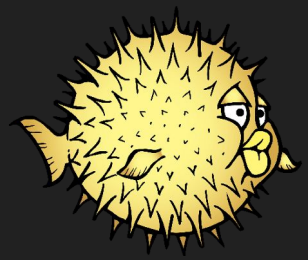
Operating Systems are actively developed! (read as = co-ops/jobs)

You can actively contribute to the open source ones now!

Linux



BSD


iOS

What is an Operating System?

Open question?

GOTO Slido.com #1993 797

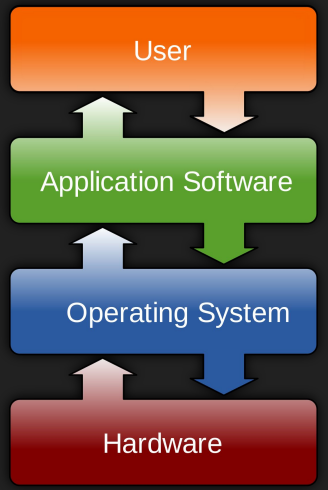
When I boot up a machine, I see Windows, Linux, or MacOS booting up, but WHAT is it doing??

What is an Operating System (OS)?

- An OS is any and all software that sits between a user program and the hardware

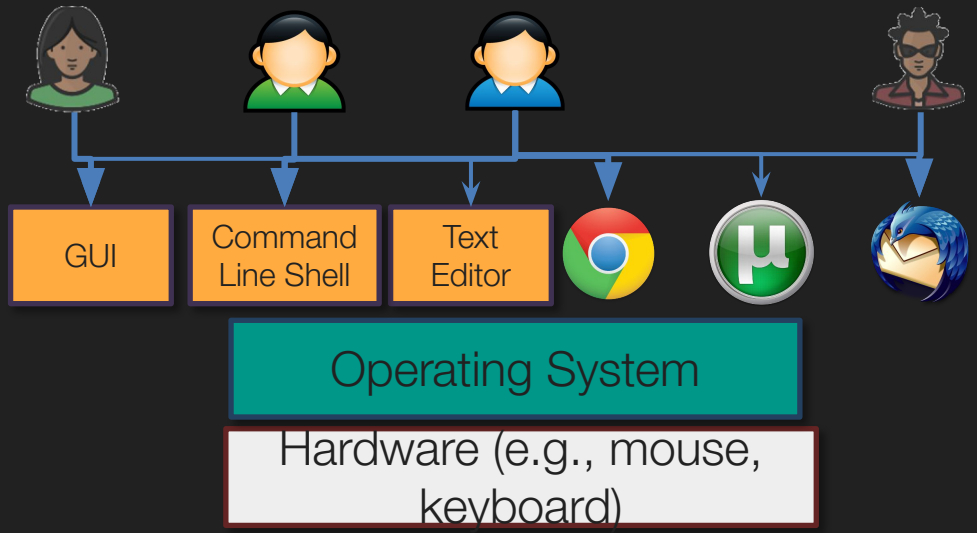
- OS is a resource manager and allocator
  - Decides between conflicting requests for hardware access
  - Attempts to be efficient and fair

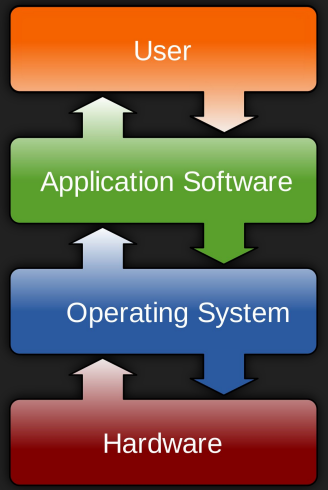
- OS is a control program
  - Controls execution of user programs
  - Prevents errors and improper use



What is an Operating System?

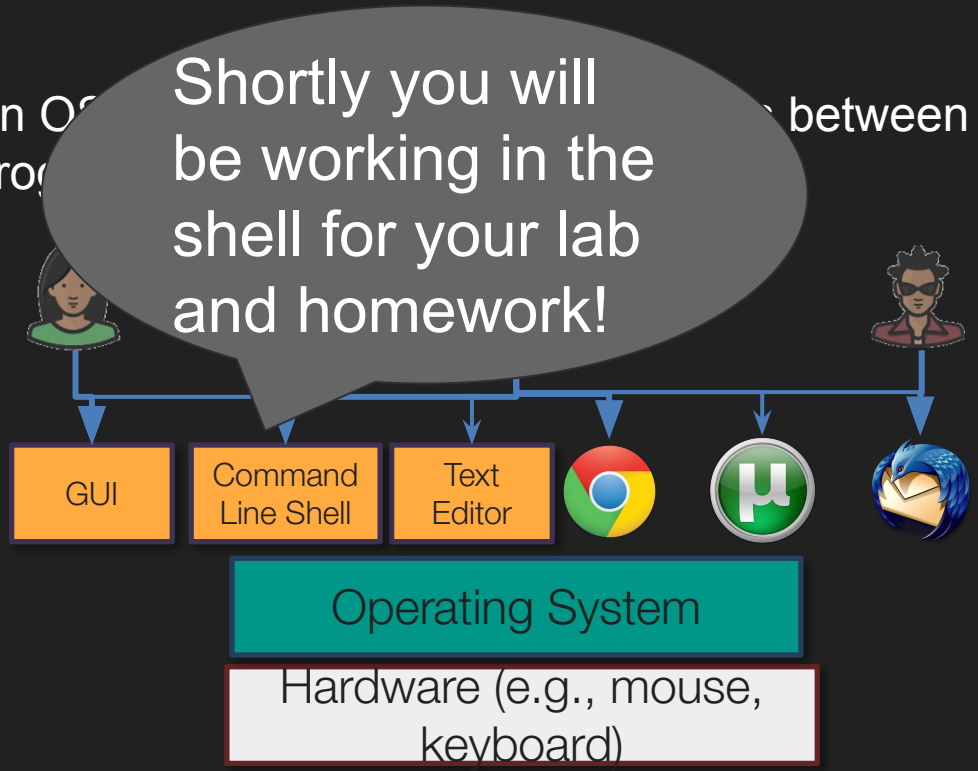
- An OS is any and all software that sits between a user program and the hardware

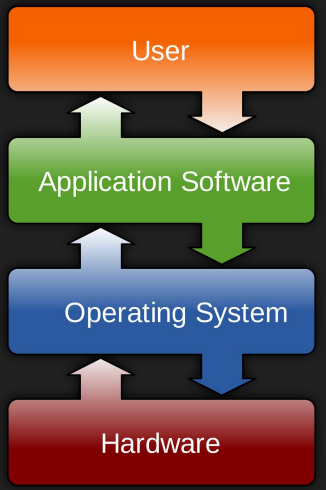


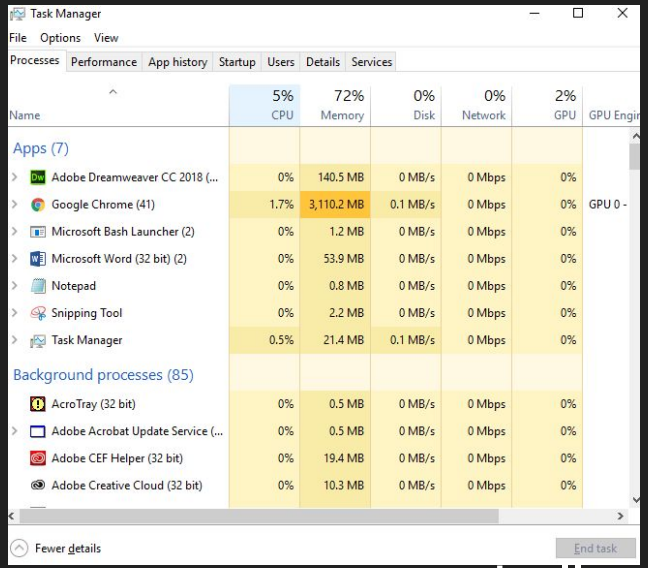


What is an Operating System?

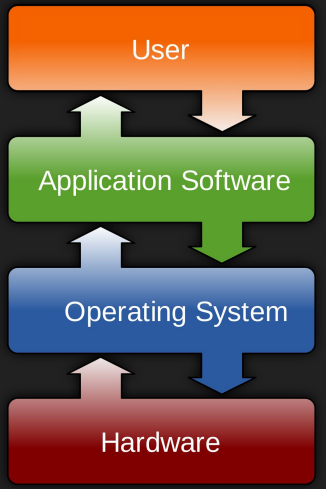
- An O… between a user prog…







- OS is a resource manager and allocator
  - Decides between conflicting requests for hardware access
  - Attempts to be efficient and fair



Two Common OS Families

- POSIX
  - Anything Unix-ish
  - e.g. Linux, BSDs, Mac, Android, iOS, QNX
- Windows
  - Stuff shipped by Microsoft

Many other operating systems may exist specific to a domain (e.g. an operating system for a car, handheld gaming device, or smart refrigerator)

Two Common OS Families

- POSIX
  - Anything Unix-ish
  - e.g. Linux, BSDs, Mac, Android, iOS
- Windows
  - Stuff shipped by Microsoft

In this course, we will work in a POSIX Environment. Our Khoury machines are Unix based.

Many other operating systems may exist specific to a domain (e.g. an operating system for a car or handheld gaming device)

Unix/Linux

[VERY] Brief History of Unix

- @AT&T Bell Labs: Ken Thompson and Dennis Ritchie (among many others)
- A response of sorts to complexity of Multics
- C developed as a higher-level language to facilitate development
- Contributed some truly elegant fundamental concepts
- Set of orthogonal tools that can act as atomic processing steps
- The pipeline and I/O redirections
- Commands are just programs
- Elegant (albeit simplistic) permission structure
- …plus many more!
- Hallmark: power and flexibility through simplicity and elegance

Who, what, why, ... Linux? https://www.linuxfoundation.org/

- Linux is a family of free open source operating systems
  - That means the code is freely available, and you can contribute to the project!
- It was created by Linus Torvalds
  - Variants of Linux are: Ubuntu, Debian, Fedora, Gentoo Linux, Arch Linux, CentOS, etc.
  - They all operate under *roughly* the same core code, which is called the kernel.
  - Often they differ by the software, user interface, and configuration settings.
  - So very often linux software for one flavor of linux will run on the other with few or no changes.
- Generally we (as systems programmers) like Linux, because it is a clean and hackable operating system.
- When many folks think of Unix-like operating systems, they may think of a hacker using a ‘command-line interface’ to program.

Over 30 years ago...

On Monday, August 26, 1991 at 2:12:08 AM UTC-4, Linus Benedict Torvalds wrote:
> Hello everybody out there using minix -
>
> I'm doing a (free) operating system (**just a hobby, won't be big and**
> **professional like gnu**) for 386(486) AT clones. This has been brewing
> since april, and is starting to get ready. I'd like any feedback on
> things people like/dislike in minix, as my OS resembles it somewhat
> (same physical layout of the file-system (due to practical reasons)
> among other things).
>
> I've currently ported bash(1.08) and gcc(1.40), and **things seem to work**.
> This implies that I'll get something practical within a few months, and
> I'd like to know what features most people would want. **Any suggestions**
> **are welcome, but I won't promise I'll implement them :-)**
>
> Linus (torv...@kruuna.helsinki.fi)
>
> PS. Yes - it's free of any minix code, and it has a multi-threaded fs.
> **It is NOT protable (uses 386 task switching etc)**, and **it probably never**
> **will support anything other than AT-harddisks, as that's all I have :-(.**

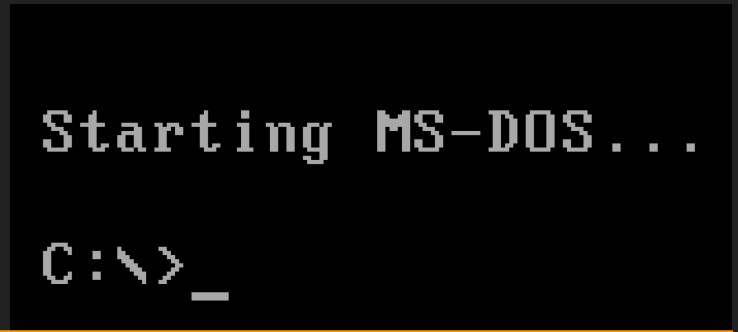
Over 30 years ago...

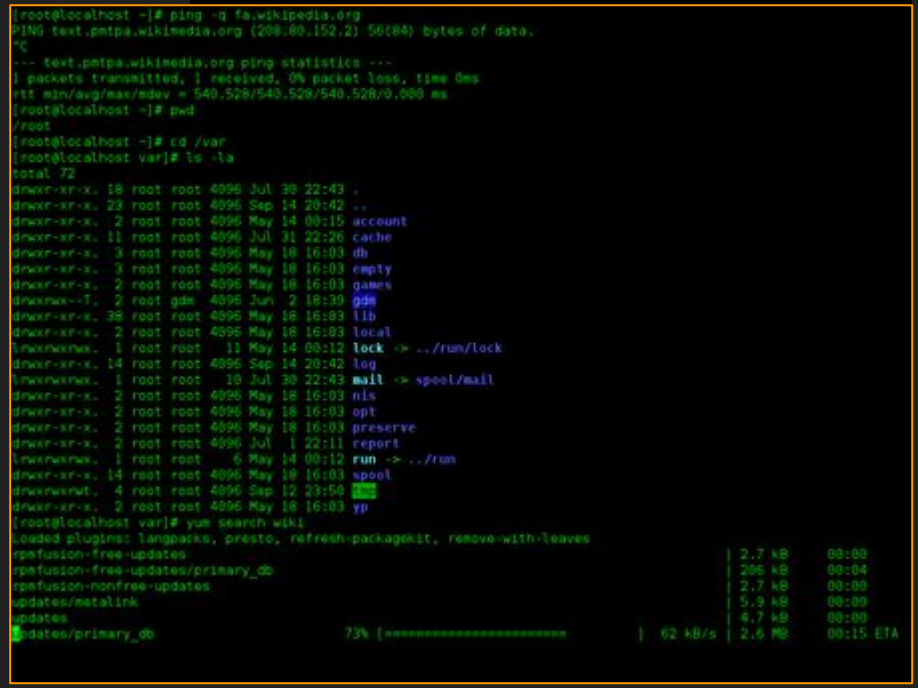
On Monday, August 26, 1991 at 2:12:08 AM UTC-4, Linus Benedict Torvalds wrote:
> Hello everybody out there using minix -
>
> I'm doing a (free) operating system (**just a hobby, won't be big and**
> **professional like gnu**) for 386(486) AT clones. This has been brewing
> since april, and is starting t
> things people like/dislike in
> (same physical layout of the f
> among other things).
>
> I've currently ported bash(1.0
> This implies that I'll get something pra ew months, and
> I'd like to know what features most n nt. **Any suggestions**
> **are welcome, but I won't promise I nt them :-)**
>
> Linus (torv...@kru sinki.fi)
>
> PS. Yes - it's free of any minix code, and it has a multi-threaded fs.
> **It is NOT protable (uses 386 task switching etc)**, and **it probably never**
> **will support anything other than AT-harddisks, as that's all I have :-(.**

Linux platforms: Alpha, ARC, ARM, ARM64, Apple M1 C6x, H8/300, Hexagon, Itanium, m68k, Microblaze, MIPS, NDS32, Nios II, OpenRISC, PA-RISC, PowerPC, RISC-V, s390, SuperH, SPARC, Unicore32, x86, x86-64, XBurst, Xtensa

The command line interface

- The command line interface is at the highest level just another program.
- Linux and Mac have terminals built-in, and Windows as well (cmd and powershell).
- From it, we can type in the names of programs to perform work for us
- (Next slide for examples)





Demonstrate simple ‘shell fu’

ls, tree, cd, ., ~

Why the command line?

- “I love GUI interfaces, so simple and sleek looking”
- Well, I will argue the command line is a lot faster than moving your mouse
- It is also very convenient for ‘scripting’ behavior that you could not so easily do in a GUI environment.
  - Executing a few commands in a row in a script is a piece of cake!
- And if you are working remotely, you often will not have any GUI environment at all!
  - (Often machines you need to access do not have a monitor attached)

Example shell script

mikeshah@DESKTOP-DDNGQVA: /mnt/c/Windows/System32

```
 1 # Lines that start with a 'hashmark' or 'pound sign'
 2 # are comments that are ignored.
 3 # You should use them liberally!
 4
 5 # This line is special and tells us we have an executable script.
 6 #!/bin/bash
 7
 8 # Output hello and two items read in as command-line arguments
 9 echo "Hello $1 $2"
10 echo "What is your age?"
11 # Read in a value
12 read myAge
13 echo "That is great you are $myAge years old!"
```

Example shell script

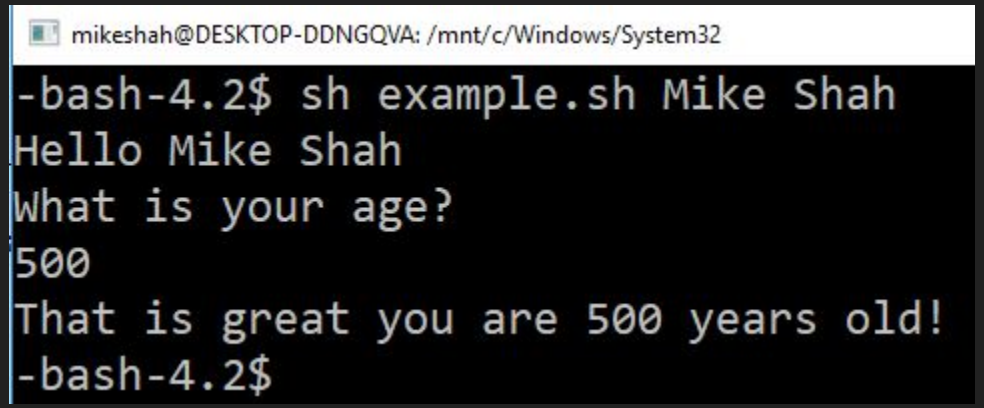
- I wrote this script in a text editor called ‘vim’
- You will have to learn VIM (or emacs) in this course.
  - It’s a great skill to have.

mikeshah@DESKTOP-DDNGQVA: /mnt/c/Windows/System32

```
 1 # Lines that start with a 'hashmark' or 'pound sign'
 2 # are comments that are ignored.
 3 # You should use them liberally!
 4
 5 # This line is special and tells us we have an executable script.
 6 #!/bin/bash
 7
 8 # Output hello and two items read in as command-line arguments
 9 echo "Hello $1 $2"
10 echo "What is your age?"
11 # Read in a value
12 read myAge
13 echo "That is great you are $myAge years old!"
```

Example shell script Executing

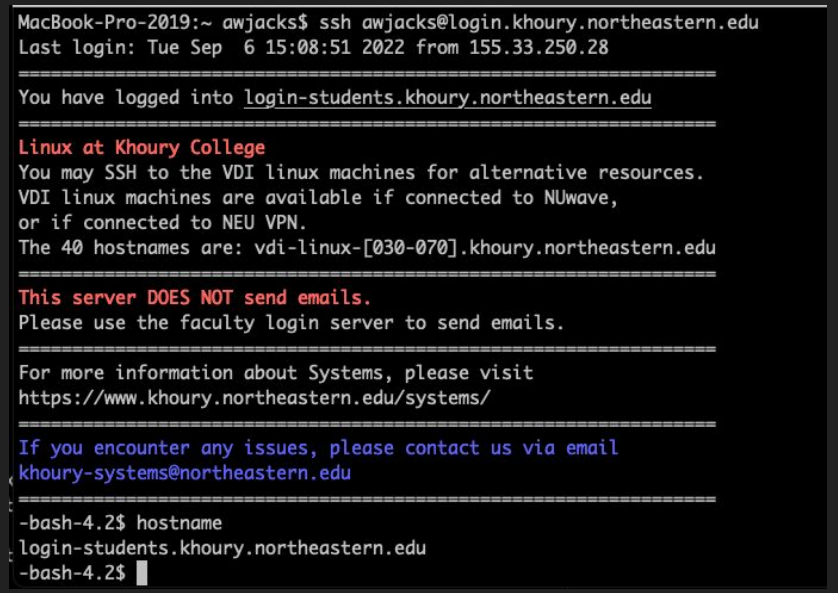
- Note “Mike Shah” are the first and second arguments passed into this program



(Am I really 500 years old? Time flies when you are having fun!)

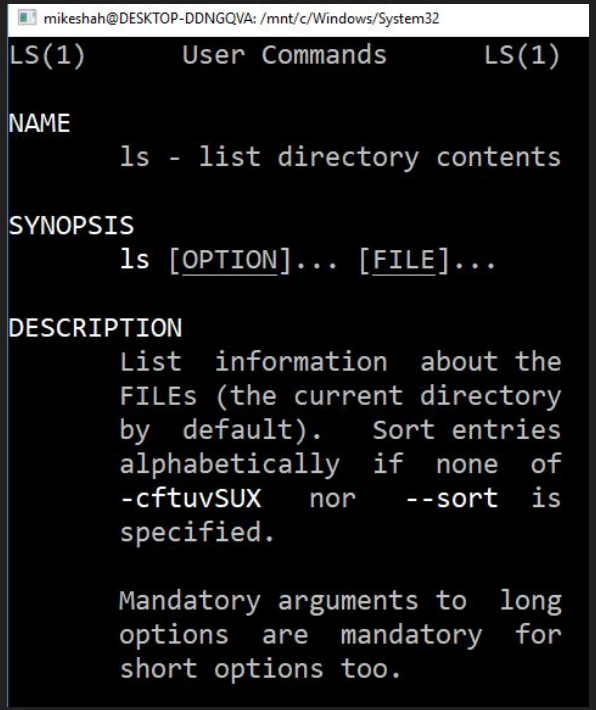
ssh - secure shell

- Our tool for remote access--which we will use for all of our work!
- `ssh Khoury_user_name@login.khoury.northeastern.edu`
- After typing in my password successfully, I am now executing commands on a machine somewhere on Northeastern's campus



Feeling overwhelmed or forgetting a command?

- Luckily there are built-in ‘manual pages’
- Called the ‘man pages’ for short.
- Simply type ‘man command_name’ for help
  - (Hit ‘q’ to quit the page when you are done)



This lecture in summary

- We are going to learn about computer systems
  - This includes software (e.g. compilers) and hardware architecture
  - Some basic operating system concepts.

- We are going to work in a Unix environment
  - This work will be performed on a command-line
  - We can access a command-line either:
    - Through SSH or a Virtual Machine

One final thing…

- Even with the best planning…
- Some things may change this semester that are beyond our control
- Everyone (including us) needs to be flexible
- If you have an issue, it is better to tell us early than at the last minute

- I’m looking forward to being your guide to Computer Systems