

NEU CS 3650 Computer Systems

Instructor: Dr. Ziming Zhao

* Acknowledgements: created based on Christo Wilson, Ferdinand Vesely, Alden Jackson, Ben Weintraub, Gene Cooperman, Peter Desnoyers' lecture slides for the same course.

First off, Logistics!

Classes will be recorded and released on Canvas
But you have to attend the class in-person

Have a notebook in front of you. Bring a laptop.

<https://khoury-cs3650.github.io/>

(Shared with other sections of CS 3650 offered by Prof. Ferdinand Vesely)

Feel free to interrupt me and ask questions.

Instructor and Teaching Assistants

Dr. Ziming Zhao

Associate Professor, Khoury College of Computer Sciences
Director, CyberspAce seCuriTy and forensIcs Lab (CactiLab)

Email: z.zhao@northeastern.edu

<http://zzm7000.github.io>

<http://cactilab.github.io>

Office hours will be T 2:00 pm - 3:30 pm or by appointment at 177
Huntington (Room 513) or the Zoom link

<https://northeastern.zoom.us/j/98423457229>

TAs: Tushin Mallick and Swadeep

Office hours: To be decided

YouTube Channel



CyberspACe securiTy and forensics lab (CactiLab)

@zimingzhao6619 296 subscribers 143 videos

CactiLab is in the Department of Computer Science and Engineering at Uni... >

Customize channel

Manage videos

HOME

VIDEOS

PLAYLISTS

COMMUNITY

CHANNELS

ABOUT



Created playlists

Sort by



2023 Spring Team Cacti Training

View full playlist



2023 Spring UB CSE 410/565 Computer Security

View full playlist



2022 Fall UB CSE 410/510 Software Security

View full playlist



2022 Spring UB CSE 410/510 Software Security

View full playlist



2021 Fall UB CSE 410/510 Software Security

View full playlist



2020 UB CSE 610 System Security - Attack and Defen...

View full playlist



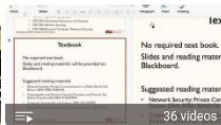
2018 ASU CSE 469 Computer and Network Forensics

View full playlist



ASU CSE 469 Computer and Network Forensics S17

View full playlist



ASU CSE 468 Computer Network Security F16

View full playlist

<https://www.youtube.com/channel/UCkSeVUu-AxytXqalx66j7Eg/playlist>

About CactiLab

Research areas:

- Systems and software security (Arm Cortex-M, Cortex-A, x86, RISC-V, FPGA, GPU, etc.)
- System problems (optimization)
- System and security for/in ML/DL/LLM
- Formally verify the security properties of crypto protocols and system code
- Hacking/CTF platforms

We need students at all levels for funded research, volunteer work, independent study, undergraduate research experience, etc.

Recruiting for eCTF 2026

2026 ECTF

[VIEW THE RULES HERE](#)

In the 2026 eCTF, teams will design and implement a secure storage solution for a chip foundry. The system must allow users with various roles to access the proper data without leaking sensitive chip designs to unauthorized parties.

Key dates:

- January 14: eCTF kickoff!
- January 31: Last day for late team registration
- February 25: Handoff
- April 15: Scoreboard closes
- April 21: Poster session
- April 24: eCTF Award Ceremony

For more information, reach us at ectf@mitre.org.



[REGISTER NOW](#)

[DOWNLOAD COMPETITION FLYER](#)

[DOWNLOAD ADVISOR INFO SHEET](#)

<https://ectf.mitre.org/>

How about you?

Take a moment and introduce yourself to someone next to you. They are going to be your colleagues for the next 14 weeks!

“e.g. What is your name? What is the worst bug you have ever encountered?”

Will your classmate(s) and you be the next:

- Jobs-Woz
- Gates-Allen
- Frances Allen
- Turing-Church
- Radhia and Patrick Cousot



Computer Systems

If algorithms and applications are the “what” of computing, then computer systems are the “how.” They deal with the mechanisms that let software run on hardware: memory management, processes, I/O, storage, scheduling, concurrency, and communication.

A “computer system” is not just hardware or software — it’s the combination of hardware, operating system, programming languages, and applications that make a computer usable.

Computer Systems

If algorithms and applications are the “what” of computing, then computer systems are the “how.” They deal with the mechanisms that let software run on hardware: memory management, processes, I/O, storage, scheduling, concurrency, and communication.

The foundation that ties together hardware and software so computers can actually run programs.

A “computer system” is not just hardware or software — it’s the combination of hardware, operating system, programming languages, and applications that make a computer usable.

Computer Systems

If algorithms and applications are the “what” of computing, then computer systems are the “how.” They deal with the mechanisms that let software run on hardware: memory management, processes, I/O, storage, scheduling, concurrency, and communication.

The foundation that ties together hardware and software so computers can actually run programs.

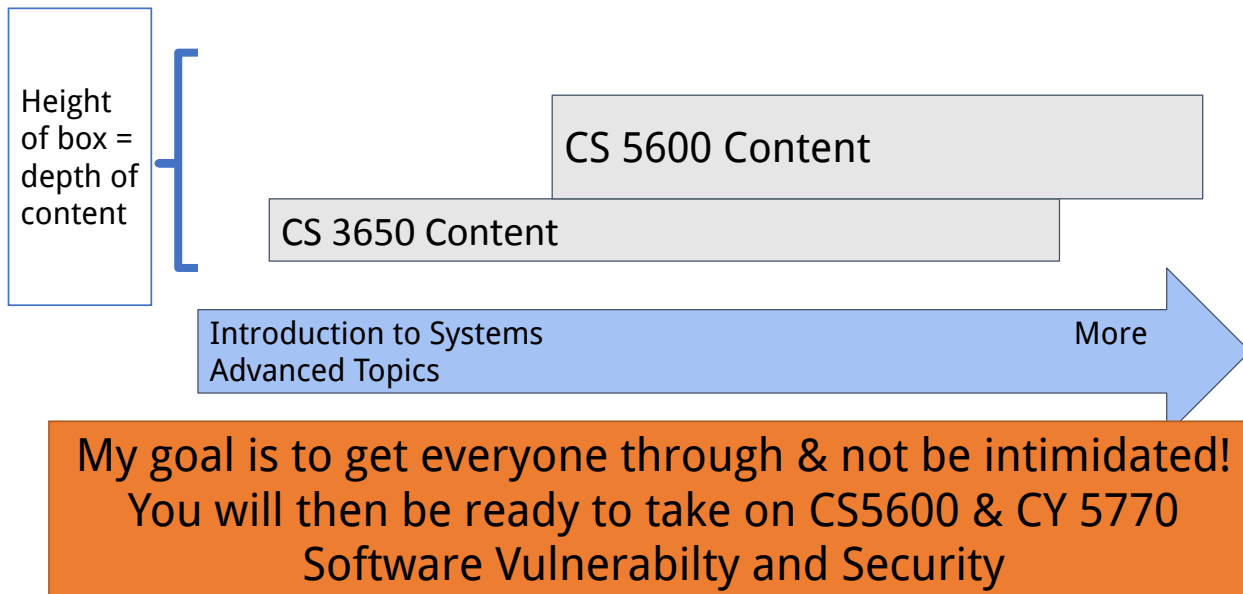
A “computer system” is not just hardware or software — it’s the combination of hardware, operating system, programming languages, and applications that make a computer usable.

Computer Systems

If algorithms and applications are the “what” of computing, then computer systems are the “how.” They deal with the mechanisms that let software run on hardware: memory management, processes, I/O, storage, scheduling, concurrency, and communication.


Computer Systems courses at Khoury College

- Two courses with the same name!
- A rough visualization of where the course is in the curriculum



Roughly Speaking this course has a few 'modules'

- Computer Systems Fundamentals
 - Terminal, C, Assembly, and Compilers
- Virtualization
 - Processes
- Computer Architecture
 - Memory/Cache/etc
- Concurrency
 - Threads/Locks/Semaphores
 - Parallelism
- Persistence
 - File Systems
 - Storage Devices
- Other Selected Topics
 - 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.

We do not have an undergraduate course entitled "Operating Systems". Only a graduate level CS 6640 – Operating Systems Implementation.

Schedule

Week (Monday)	Topics	Assignments & Labs
1 (Sep 1)	Intro to Computer Systems	<input type="checkbox"/> Assignment 1 out
2 (Sep 8)	Assembly	<input type="checkbox"/> A1 due <input type="checkbox"/> Assignment 2 out
3 (Sep 15)	Memory, the Stack, Recursion	<input type="checkbox"/> A2 due <input type="checkbox"/> Assignment 3 out
4 (Sep 22)	Intro to C	<input type="checkbox"/> A3 due <input type="checkbox"/> Assignment 4 out
5 (Sep 29)	Processes	<input type="checkbox"/> A4 due <input type="checkbox"/> Project 1 out
6 (Oct 6)	File I/O	<input type="checkbox"/> Exam 1 (10/16) <input type="checkbox"/> Project 1 due <input type="checkbox"/> Assignment 5 out
7 (Oct 13)	Virtual Memory	
8 (Oct 20)	Concurrency	<input type="checkbox"/> A5 due <input type="checkbox"/> Assignment 6 out
9 (Oct 27)	Concurrency	<input type="checkbox"/> A6 due <input type="checkbox"/> Assignment 7 out
10 (Nov 3)	OS Kernels, Booting, xv6	<input type="checkbox"/> A7 due <input type="checkbox"/> Assignment 8 out
11 (Nov 10)	OS Kernels, Booting, xv6	<input type="checkbox"/> A8 due
12 (Nov 17)	File Systems	<input type="checkbox"/> Exam 2 (11/20) <input type="checkbox"/> Project 2 out
13 (Nov 24)	File Systems	
13 (Dec 1)	Wrap-up	<input type="checkbox"/> Project 2 due

I will be out of town. Will be an online class and Exam 1

Why using 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 however—through the creativity and the art of computer science.
- Computer Science is an art!

"No more magic"

- We do not have to look at machines any more and think there is magic going on.
- Instead, we want to understand the inner working
- Someone programmed our operating systems, devices, and software
 - And they started off where you are!



webs



The Power of Abstraction



- Abstraction enables us to manage complexity by hiding details and exposing only essential features.
- It supports reasoning at the right level without being overwhelmed by low-level implementation.

<https://www.youtube.com/watch?v=qAKrMdUycb8>

The Power of Abstraction

"Any problem in computer science can be solved by another level of indirection,"

– [David Wheeler](#) and [Butler Lampson](#)

This is a guiding principle stating that introducing a layer of abstraction or indirection can simplify complex problems by breaking them down or decoupling components

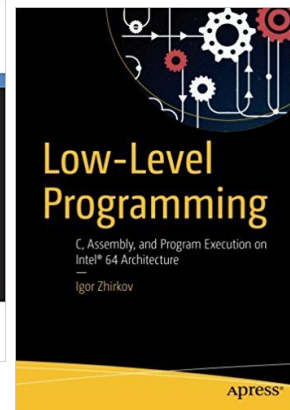
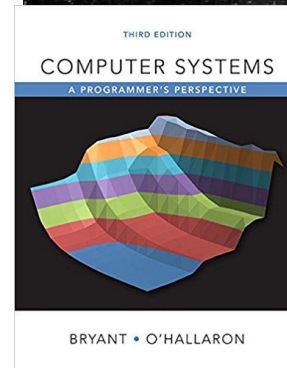
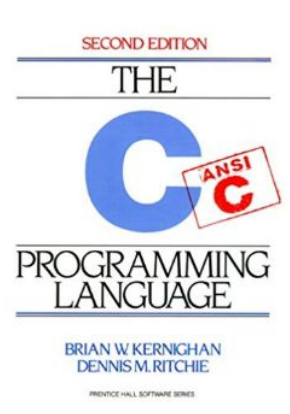
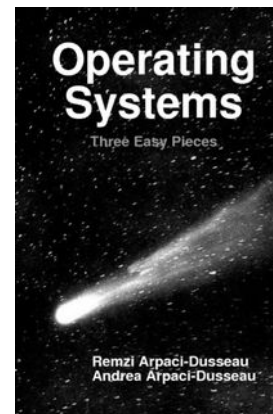
Course Goals

By the end of this course, you will:

- Have a working knowledge of C
- Be comfortable working at the terminal prompt in a Unix environment
- Build and use tools for inspecting and debugging programs at a low level
- Be comfortable with concepts like concurrency and parallelism
- Have a better understanding of the basics and internals of operating systems
- Gain some experience working on a large scale codebase
- Position yourself for jobs as a systems programmer

Books

- Main
 - [Dive Into Systems](#)
 - [Operating Systems: Three Easy Pieces \(aka OSTEP\)](#)
- Additional
 - [MIT PDOS xv6 x86 32bit source code](#)
 - [xv6 book x86 edition](#)
- Recommended
 - Low-Level Programming: C, Assembly, and Program Execution on Intel® 64 Architecture
 - C Programming Language Book
 - [Computer Systems: A Programmer's Perspective, 3 Edition](#)



Course Resources and Website

<https://khoury-cs3650.github.io/>

Shared with other sections of CS 3650 offered by Prof. Ferdinand Vesely

All class materials can be found on the website

Learning systems we will use: Canvas, Piazza, gradescope, GitHub

Course Resources and Website

<https://khoury-cs3650.github.io/>

CS3650 Computer Systems

[General](#)[Syllabus](#)[Schedule](#)[Office Hours](#)[Assignments](#)[Resources](#)

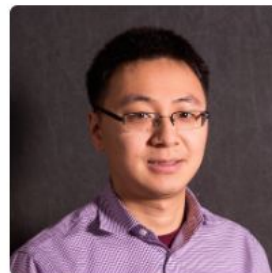
General Information

- [Instructor](#)
- [Location and Time](#)
- [Teaching Assistants](#)

Instructor



Ferdinand Vesely
f.vesely@northeastern



Ziming Zhao
z.zhao@northeastern

Location and Time

Section	Instructor	Times	Location
1	F. Vesely	Mon/Wed/Thu 1:35-2:40pm	Churchill Hall 103
2	Z. Zhao	Tue 11:45am-1:25pm, Thu 2:50-4:30pm	Shillman Hall 420
3	F. Vesely	Mon/Wed/Thu 4:35-5:40pm	Churchill Hall 103

Teaching Style

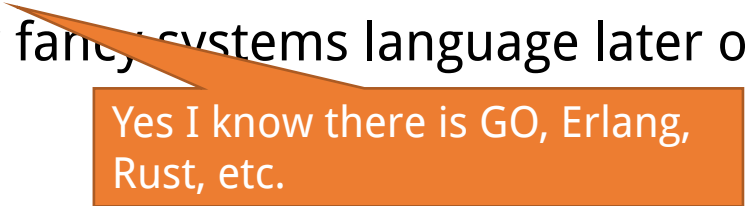
- Everyone learns differently--optimize as needed along the way
 - There will be lectures (for auditory learners) - Code reading and demos
 - Many visuals on slides (for visual learners)
 - Labs and assignments (for kinesthetic learners)
- This is a very hands-on class, we will build things
- There will be plenty of opportunity to make mistakes
Do not be afraid to be wrong
 - The worst-case scenario is we review
- Do ask questions!
 - I try to avoid randomly calling on students--but do participate!
- Come to office hours! Mine or the TAs or both!

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**)
- But do not reveal solutions

Expectations

- You have taken some 'programming' related class.
 - In the instance that you have not--you can still perform well.
 - i.e. Make sure you do the readings
- You know at least one programming language well
- In this course we will use C and get exposed to x86 32bit and 64bit assembly
 - C is (still) the industry standard
 - (You can pick up whatever other fancy systems language later once you learn one)



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)

Grades

To calculate final grades, we simply sum up the points obtained by each student (the points will sum up to some number x out of 100, rounded up for fractions) and then use the following scale to determine the letter grade:

Numeric Grade	Letter Grade
[0-59]	F
[60-62]	D-
[63-66]	D
[67-69]	D+
[70-72]	C-
[73-76]	C
[77-79]	C+
[80-82]	B-
[83-86]	B
[87-89]	B+
[90-92]	A-
[93-100]	A

We do not curve the grades in any way.

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

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

Course Materials/Equipment

- A laptop
- The laptop's operating system shouldn't matter, however, having a Unix-like environment (Linux, macOS, WSL, *BSD, ...) is an advantage
- We will provide you with a cloud-based Virtual Machine for you to work on
- We will use Linux for most of the course
- With any programming assignment, the assumption is, that you have tested your code **on the provided VM**. Regrade requests based on "it worked on my machine" will be rejected

Requests for Regrading

After grades have been posted, there is a **3** day window to request a regrade from the TA. If you have further issues with TA's regrade, you may challenge the regrade with a professor.

Academic Integrity

Your first assignment is to 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 friends

Academic Integrity

- Discussion is encourage. But, you cannot share your code, exploits to your classmates or post them online.
- The university, college, and department policies against academic dishonesty will be strictly enforced. To understand your responsibilities as a student read: [NEU 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** or **>F<** on the final grade. And, the violation will be reported to the Dean's office.

Disability Access Services

If you need DAS, please inform me in the first two weeks.

How to be successful in CS 3650

- Read the assigned reading before class
- Attend the class
 - Ask questions
 - Answer questions
- You need theoretical backgrounds 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 expectation
 - Debugging could take forever

Questions?

Here is what 'C' looks like

```
#include <stdio.h>

int main(){

    puts("Hello Computer Systems!\n");

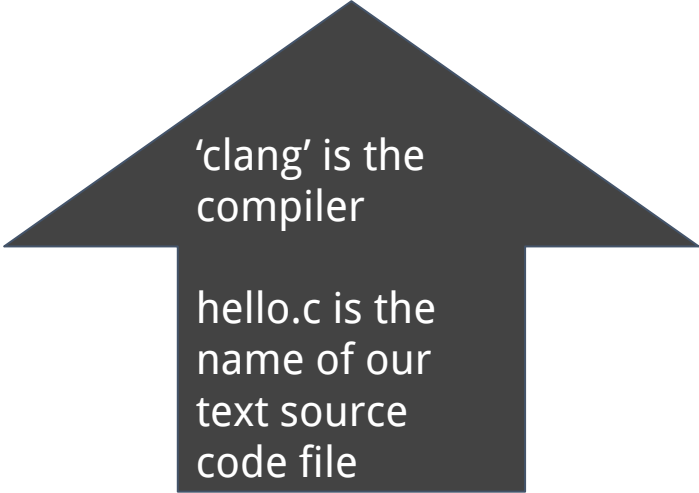
    return 0;
}
```

Here is what 'C' looks like

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

Here is what 'C' looks like

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

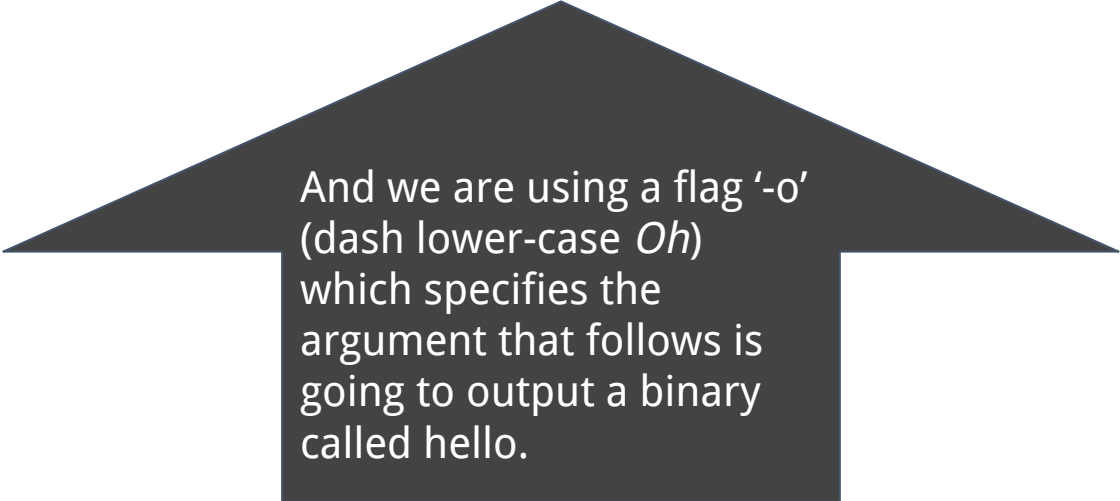


'clang' is the
compiler

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

Here is what 'C' looks like

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



And we are using a flag '-o' (dash lower-case *Oh*) which specifies the argument that follows is going to output a binary called hello.

Here is what 'C' looks like

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

```
1 #include <stdio.h>
2
3 int main(){
4
5     puts("Hello Computer System!");
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: `clang hello.c -o hello`


```
1 #include <stdio.h>
2
3 int main(){
4
5     puts("Hello Computer");
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: `clang hello.c -o hello`

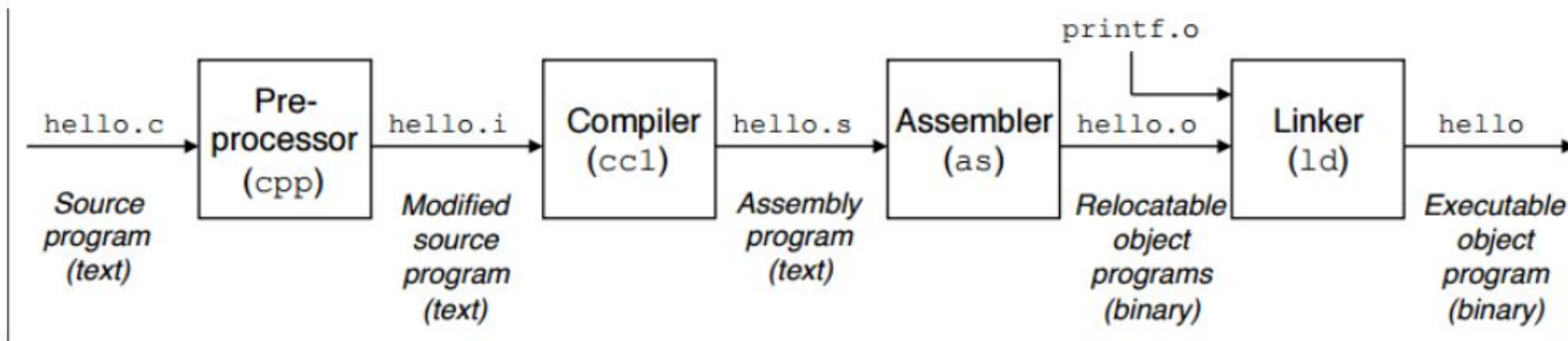
```
1 #include <stdio.h>
2
3 int main(){
4
5     puts("Hello Computer Science");
6
7     return 0;
8 }
```



And finally we are done with our program and we **return**.

C and the compilation process

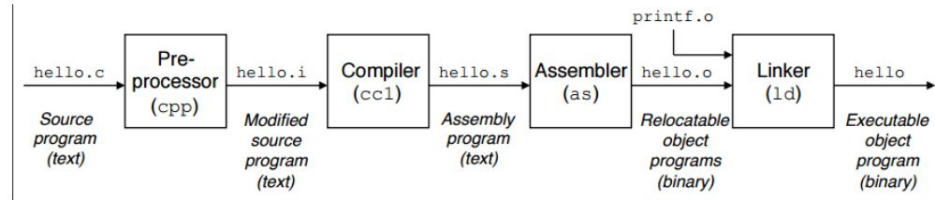
- In a picture, this is the compilation process from start to finish
- (Note in this class we'll use clang, but gcc is also fine)



```
clang -Wall -save-temps hello.c -o hello
```


Little exercise to see what compiler is doing

- Generate assembly code
 - `clang -S hello.c`
- Investigate assembly
- Compile assembly to executable file
 - `clang hello.s -o hello`
- Generate Object file
 - `clang -c hello.s`
- View Object File
 - `nl hello.o` (unreadable)
- Investigate Object File
 - `objdump -d hello.o`
(disassembly – shows assembly of machine instructions)
 - `objdump -t hello.o` (shows symbol table)



Quick view of the assembly

clang -S hello.c

cat hello.s

```
→ hello-CS clang -S hello.c
→ hello-CS ls
hello hello.c hello.s
→ hello-CS cat hello.s
        .text
        .file    "hello.c"
        .globl   main
        .p2align 4, 0x90
        .type    main,@function
main:
        .cfi_startproc
# %bb.0:
        pushq    %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq     %rsp, %rbp
        .cfi_def_cfa_register %rbp
        subq     $16, %rsp
        movl     $0, -4(%rbp)
        leaq     .L.str(%rip), %rdi
        callq    puts@PLT
        xorl     %eax, %eax
        addq     $16, %rsp
        popq     %rbp
        .cfi_def_cfa %rsp, 8
        retq
.Lfunc_end0:
        .size    main, .Lfunc_end0-main
        .cfi_endproc
# -- End function
        .type    .L.str,@object
        .section .rodata.str1.1,"aMS",@progbits,1
.L.str:
        .asciz   "Hello Computer Systems!"
        .size    .L.str, 24

.ident    "Ubuntu clang version 16.0.0 (++20230112052731+edba5d58cd19-1-exp1~20230112172830.506)"
.section   ".note.GNU-stack","",@progbits
.addrsig
.addrsig_sym puts
```

Quick view of objdump

objdump -d ./hello

```
hello-CS objdump -d ./hello
./hello:      file format elf64-x86-64

Disassembly of section .init:

0000000000001000 <.init>:
1000:    f3 0f 1e fa                endbr64
1004:    48 83 ec 08                sub    $0x8,%rsp
1008:    48 8b 05 d9 2f 00 00      mov    0x2fd9(%rip),%rax        # 3fe8 <__gmon_start__>
100f:    48 85 c0                  test   %rax,%rax
1012:    74 02                     je     1016 <_init+0x16>
1014:    ff d0                     callq  *%rax
1016:    48 83 c4 08                add    $0x8,%rsp
101a:    c3                       retq

Disassembly of section .plt:

0000000000001020 <.plt>:
1020:    ff 35 e2 2f 00 00      pushq 0x2fe2(%rip)        # 4008 <GLOBAL_OFFSET_TABLE.+0x8>
1026:    ff 25 e4 2f 00 00      jmpq   *0x2fe4(%rip)        # 4010 <GLOBAL_OFFSET_TABLE.+0x10>
102c:    0f 1f 40 00             nopl    0x0(%rax)

0000000000001030 <puts@plt>:
1030:    ff 25 e2 2f 00 00      jmpq   *0x2fe2(%rip)        # 4018 <puts@GLIBC_2.2.5>
1036:    68 00 00 00 00 00      pushq  $0x0
103b:    e9 e0 ff ff             jmpq   1020 <.plt>

Disassembly of section .plt.got:

0000000000001040 <__cxa_finalize@plt>:
1040:    ff 25 b2 2f 00 00      jmpq   *0x2fb2(%rip)        # 3ff8 <__cxa_finalize@GLIBC_2.2.5>
1046:    66 90                     xchg    %ax,%ax

Disassembly of section .text:

0000000000001050 <_start>:
1050:    f3 0f 1e fa                endbr64
1054:    31 ed                     xor     %ebp,%ebp
1056:    49 89 d1                  mov     %rdx,%r9
1059:    5e                       pop     %rsi
105a:    48 89 e2                  mov     %rsp,%rdx
105d:    48 83 e4 f0              and     $0xfffffffffffffff0,%rsp
1061:    50                       push    %rax
1062:    54                       push    %rsp
1063:    4c 8d 05 76 01 00 00     lea     0x176(%rip),%r8        # 11e0 <__libc_csu_fini>
106a:    48 8d 0d ff 00 00 00     lea     0xff(%rip),%rcx        # 1170 <__libc_csu_init>
1071:    48 8d 3d c8 00 00 00     lea     0xc8(%rip),%rdi        # 1140 <main>
1078:    ff 15 62 2f 00 00      callq   *0x2f62(%rip)        # 3fe0 <__libc_start_main@GLIBC_2.2.5>
107e:    f4                       hlt
107f:    90                       nop
```

Quick view of objdump

objdump -t ./hello

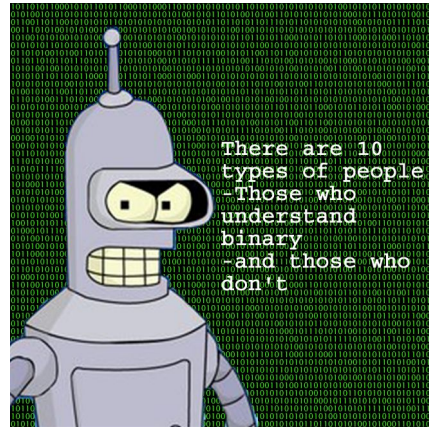
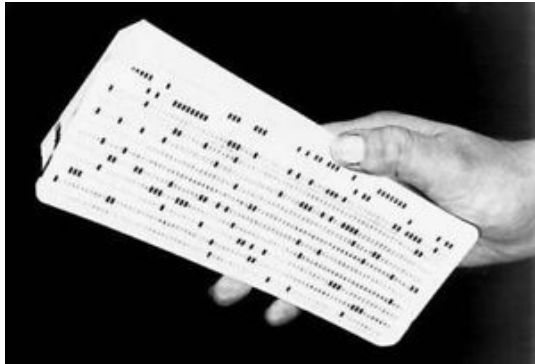
```
+ hello-CS objdump -t ./hello

./hello:      file format elf64-x86-64

SYMBOL TABLE:
00000000000002a8 l d .interp 0000000000000000 .interp
00000000000002c4 l d .note.gnu.build-id 0000000000000000 .note.gnu.build-id
00000000000002e8 l d .note.ABI-tag 0000000000000000 .note.ABI-tag
0000000000000308 l d .gnu.hash 0000000000000000 .gnu.hash
0000000000000338 l d .dynsym 0000000000000000 .dynsym
00000000000003d8 l d .dynstr 0000000000000000 .dynstr
000000000000045a l d .gnu.version 0000000000000000 .gnu.version
0000000000000468 l d .gnu.version_r 0000000000000000 .gnu.version_r
0000000000000488 l d .rela.dyn 0000000000000000 .rela.dyn
0000000000000548 l d .rela.plt 0000000000000000 .rela.plt
0000000000001000 l d .init 0000000000000000 .init
0000000000001028 l d .plt 0000000000000000 .plt
0000000000001048 l d .plt.got 0000000000000000 .plt.got
0000000000001058 l d .text 0000000000000000 .text
00000000000011e8 l d .fini 0000000000000000 .fini
0000000000002000 l d .rodata 0000000000000000 .rodata
0000000000002020 l d .eh_frame_hdr 0000000000000000 .eh_frame_hdr
0000000000002060 l d .eh_frame 0000000000000000 .eh_frame
0000000000003de8 l d .init_array 0000000000000000 .init_array
0000000000003df8 l d .fini_array 0000000000000000 .fini_array
0000000000003df8 l d .dynamic 0000000000000000 .dynamic
0000000000003f8 l d .got 0000000000000000 .got
0000000000004000 l d .got.plt 0000000000000000 .got.plt
0000000000004020 l d .data 0000000000000000 .data
0000000000004038 l d .bss 0000000000000000 .bss
0000000000000000 l d .comment 0000000000000000 .comment
0000000000000000 l df *ABS* 0000000000000000 crtstuff.c
0000000000001080 l F .text 0000000000000000 deregister_tm_clones
00000000000010b0 l F .text 0000000000000000 register_tm_clones
00000000000010f0 l F .text 0000000000000000 __do_global_dtors_aux
0000000000004030 l O .bss 0000000000000001 completed.8061
0000000000003df8 l O .fini_array 0000000000000000 __do_global_dtors_aux_fini_array_entry
0000000000001138 l F .text 0000000000000000 frame_dummy
0000000000003de8 l O .init_array 0000000000000000 __frame_dummy_init_array_entry
0000000000000000 l df *ABS* 0000000000000000 hello.c
0000000000000000 l df *ABS* 0000000000000000 crtstuff.c
0000000000000214c l O .eh_frame 0000000000000000 __FRAME_END__
0000000000000000 l df *ABS* 0000000000000000
0000000000003df8 l .init_array 0000000000000000 __init_array_end
0000000000003df8 l O .dynamic 0000000000000000 __DYNAMIC
0000000000003de8 l .init_array 0000000000000000 __init_array_start
0000000000002020 l .eh_frame_hdr 0000000000000000 __GNU_EH_FRAME_HDR
0000000000004000 l O .got.plt 0000000000000000 __GLOBAL_OFFSET_TABLE__
0000000000001000 l F .init 0000000000000000
00000000000011e8 g F .text 0000000000000005 _libc_csu_fini
0000000000000000 w *UND* 0000000000000000 __ITM_deregisterTMCloneTable
0000000000004020 w .data 0000000000000000 data_start
0000000000000000 F *UND* 0000000000000000 puts@GLIBC_2.2.5
0000000000004030 g .data 0000000000000000 _edata
00000000000011e8 g F .fini 0000000000000000 _hidden_fini
0000000000000000 F *UND* 0000000000000000 __libc_start_main@GLIBC_2.2.5
0000000000004020 g .data 0000000000000000 __data_start__
0000000000000000 w *UND* 0000000000000000 __gmon_start__
0000000000004020 g O .data 0000000000000000 _hidden __dso_handle
0000000000002000 g O .rodata 0000000000000004 __IO_stdin_used
0000000000001170 g F .text 0000000000000065 _libc_csu_init
0000000000004038 g .bss 0000000000000000 __end
0000000000001050 g F .text 000000000000002f __start
0000000000004030 g .bss 0000000000000000 __bss_start
0000000000001148 g F .text 0000000000000023 main
0000000000004030 g O .data 0000000000000000 _hidden __TMC_END__
0000000000000000 w *UND* 0000000000000000 __ITM_registerTMCloneTable
0000000000000000 w F *UND* 0000000000000000 __cxa_finalize@GLIBC_2.2.5
```

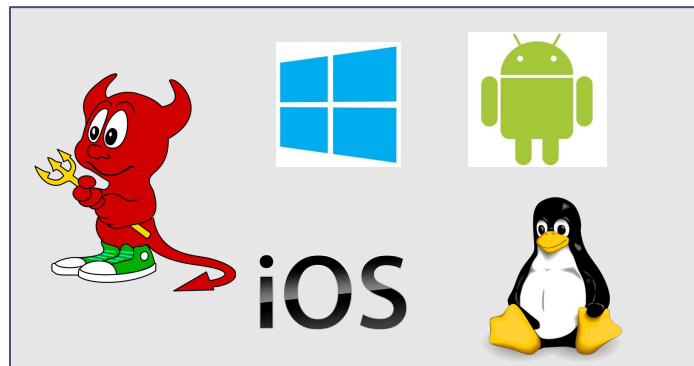
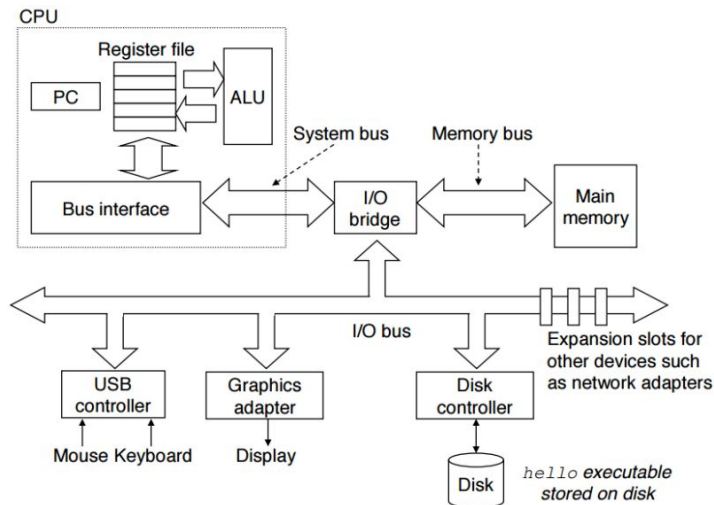
So compilers are pretty neat

- 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!)



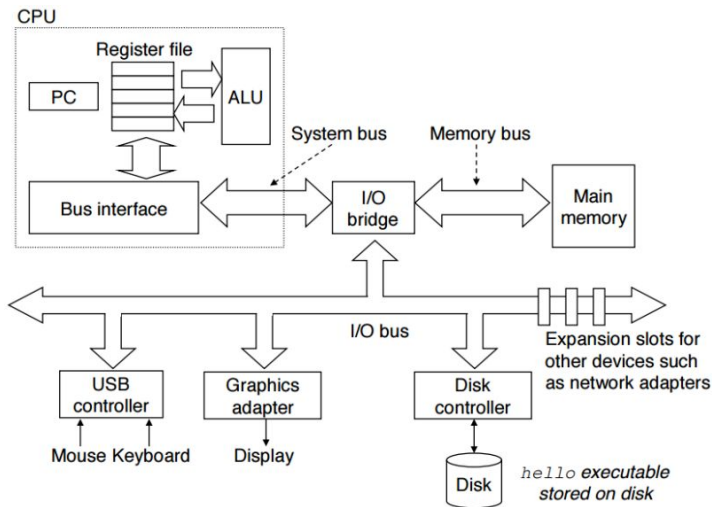
C and compilers allow us to control the system

- Core pieces of systems include hardware(left) and operating system (right)



C and compilers allow us to control the system

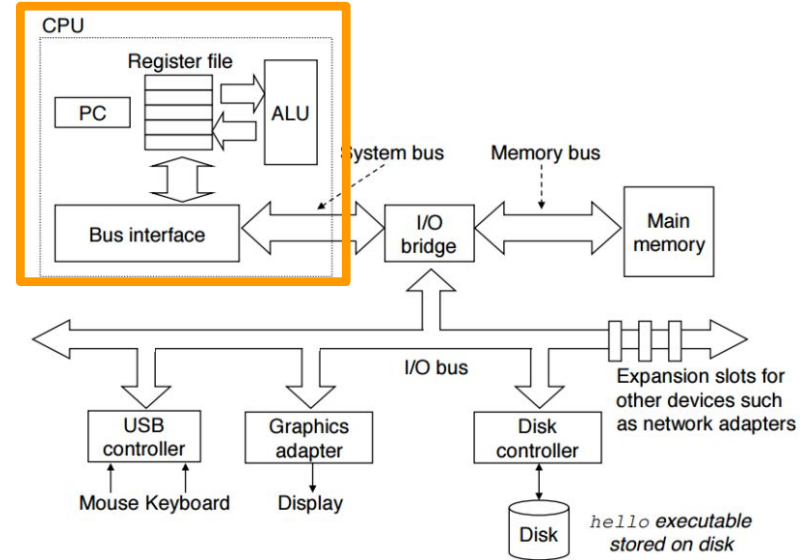
- Core pieces of systems include hardware(left) and operating system (right)



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

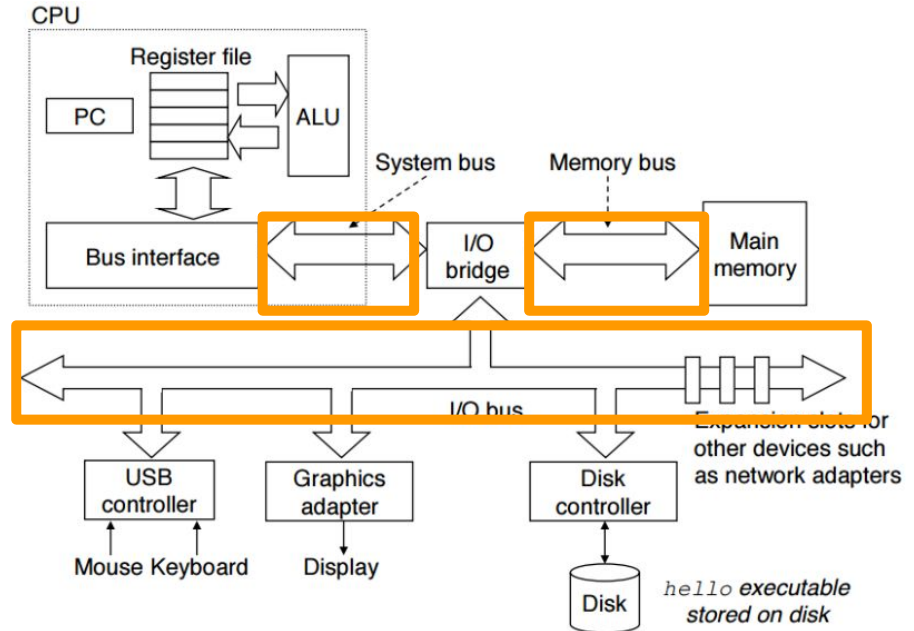
Modern Hardware Visual Abstraction

- CPU is the “brain” of modern hardware
 - That’s where 1 instruction is executed at a time
 - Only 1!
 - (Note: Modern computers have multiple cores/hardware threads)
- 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.

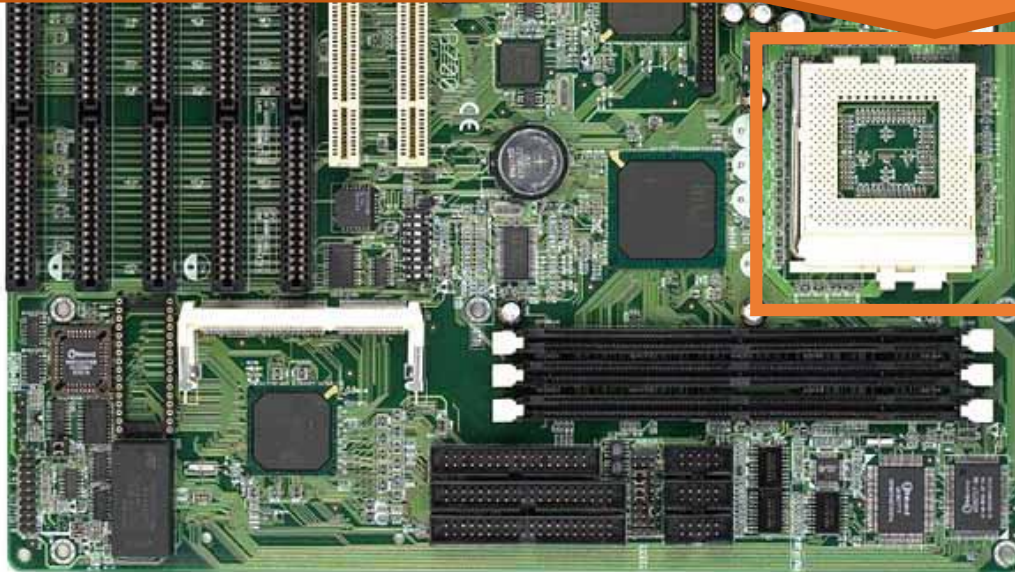


Modern Hardware Visual Abstraction

- 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.

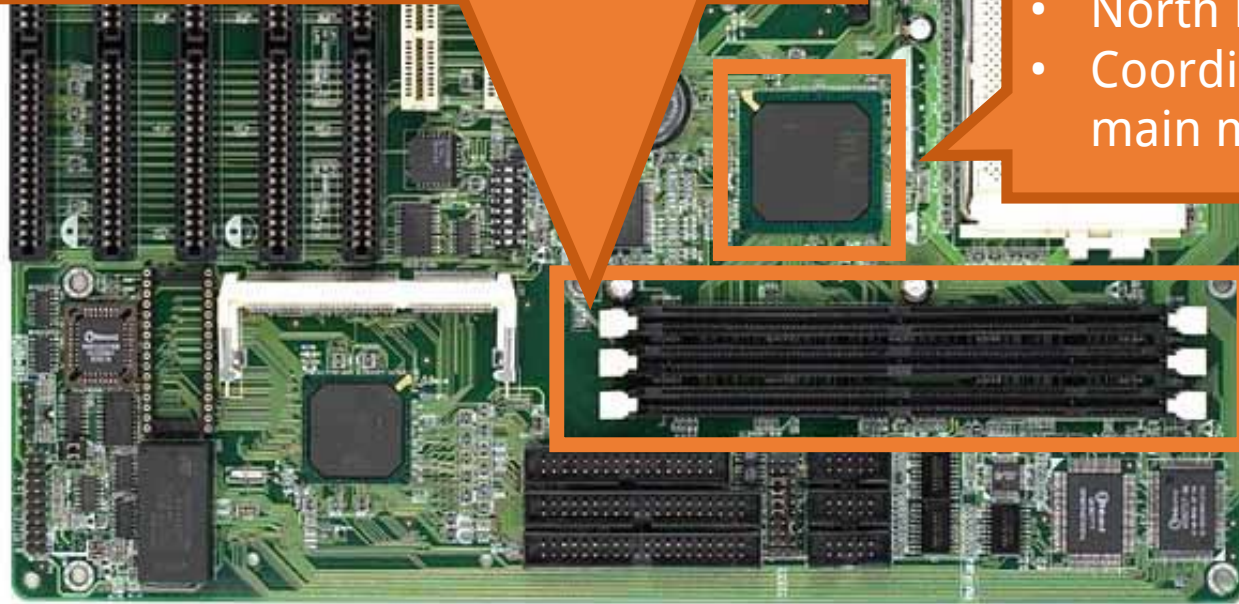


- CPU Socket
- Many different physical socket standards
 - This a Pentium 1 socket
- Physical standard is less important than Instruction Set Architecture (ISA)
 - IBM PCs are Intel 80386 compatible
 - Original x86 design
 - Intel, AMD, VIA
- Today's dominant ISA: x86-64, developed by AMD



- Slots for random access memory (RAM)
- Pre-1993: DRAM (Dynamic RAM)
- Post-1993: SDRAM (Synchronous DRAM)
- Current standard: Double data rate SDRAM (DDR SDRAM)

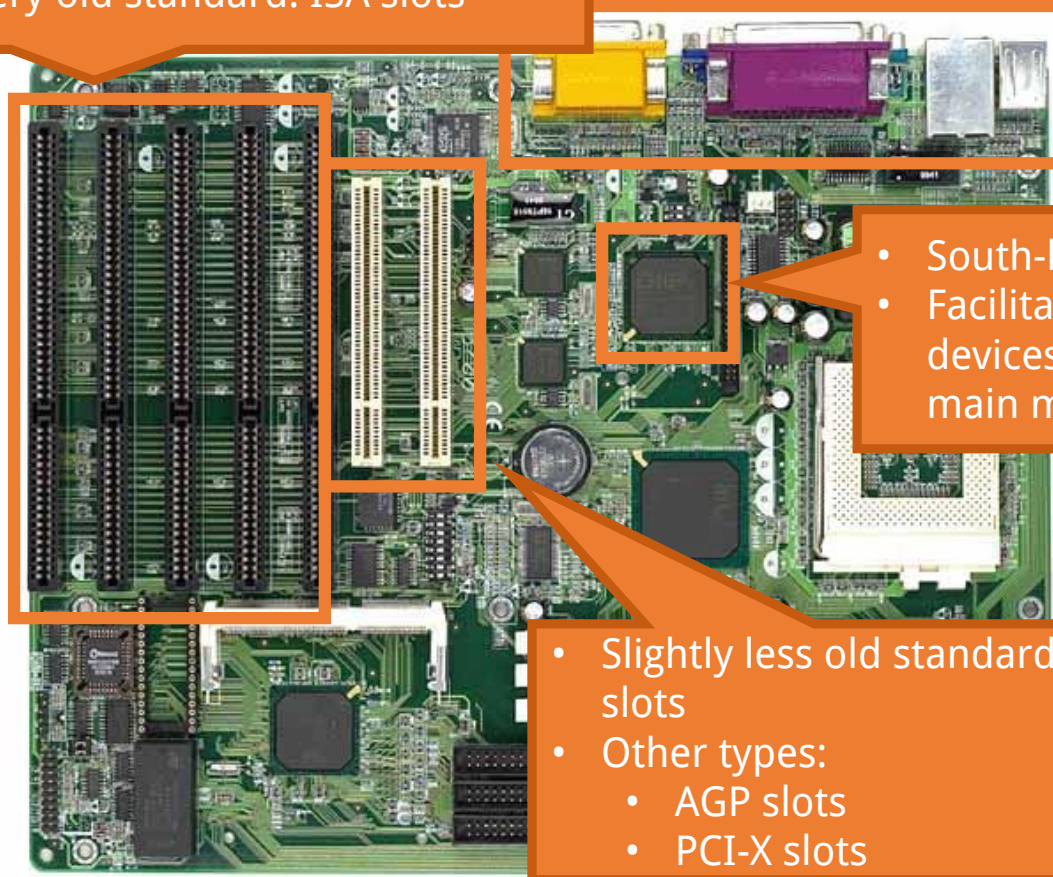
- North Bridge
- Coordinates access to main memory



The northbridge was a dedicated chip that connected the CPU to fast components like RAM and GPU. It no longer exists as a separate chip because its functions are now inside modern CPUs.

- I/O device slots
- Attached to the south-bridge bus
- Very old standard: ISA slots

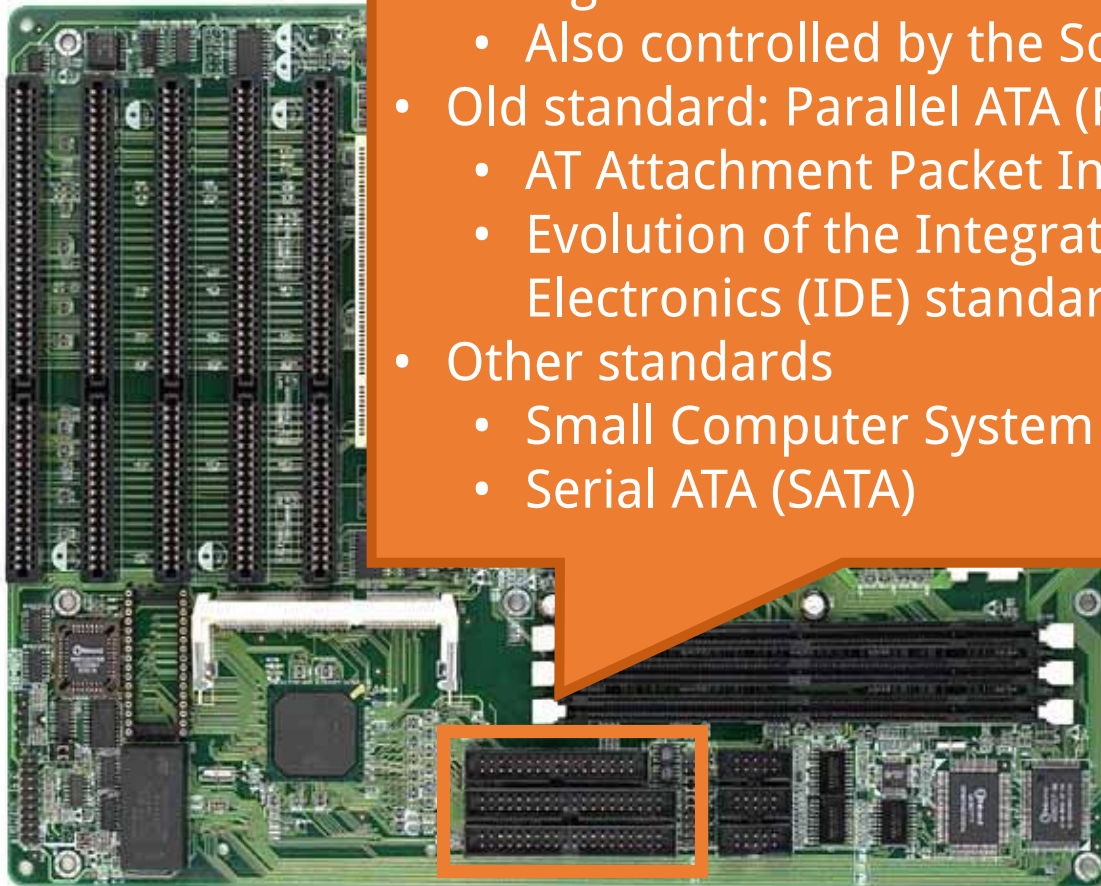
- Built in I/O also on the PCI/ISA bus



- South-bridge
- Facilitates I/O between devices, the CPU, and main memory

- Slightly less old standard: PCI slots
- Other types:
 - AGP slots
 - PCI-X slots

The south bridge is the other half of the traditional computer chipset, handling slower-speed I/O (input/output) operations and peripheral connections.



- Storage connectors
 - Also controlled by the South Bridge
- Old standard: Parallel ATA (P-ATA)
 - AT Attachment Packet Interface (ATAPI)
 - Evolution of the Integrated Drive Electronics (IDE) standard
- Other standards
 - Small Computer System Interface (SCSI)
 - Serial ATA (SATA)

PCI slot

PCI-x16 slots

USB
Headers

North
Bridge

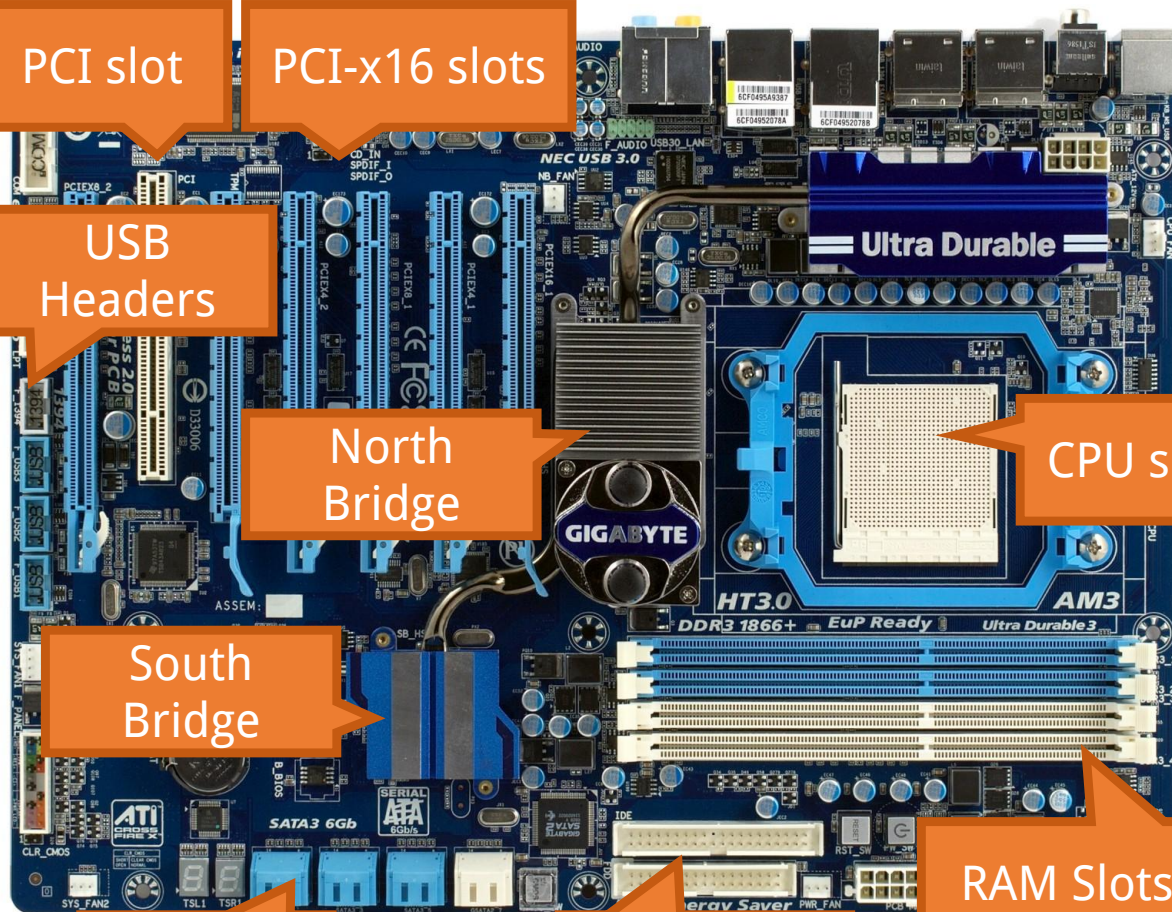
CPU socket

South
Bridge

RAM Slots

SATA
Plugs

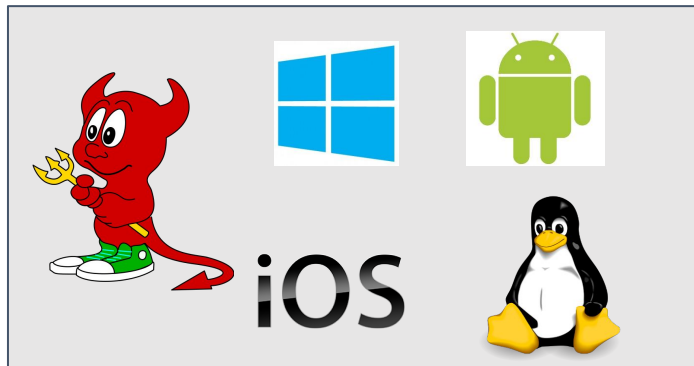
PATA
Connectors



C and compilers allow us to control the system

- Core pieces of systems include hardware(left) and operating system (right)

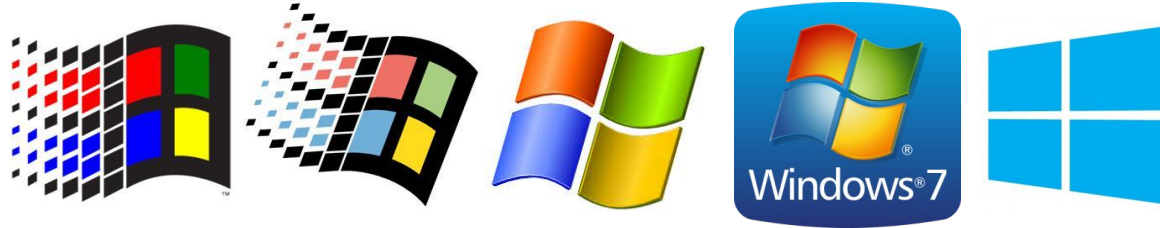
Let's take a moment to think about operating systems



What is an Operating System?

Many Different OSES

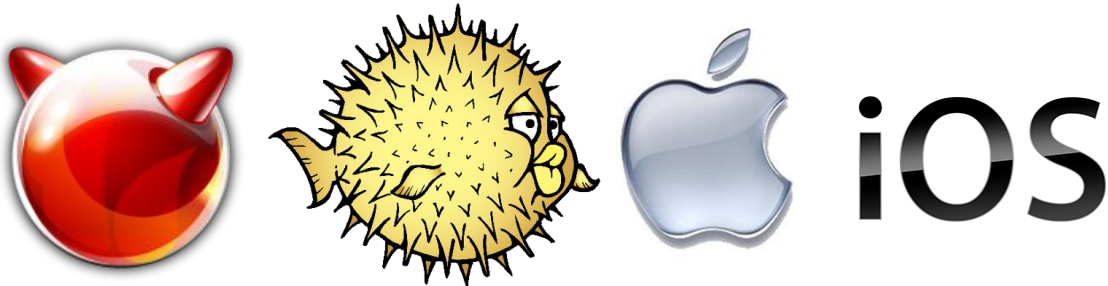
Windows



Linux



BSD



iOS

Many Different OSeS

Windows



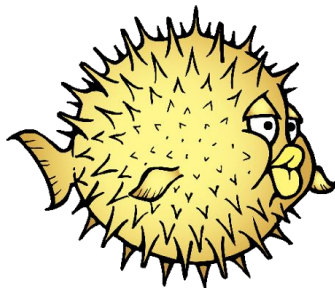
Operating Systems are actively
developed!

Linux

You can actively contribute to the open
source ones now!



BSD



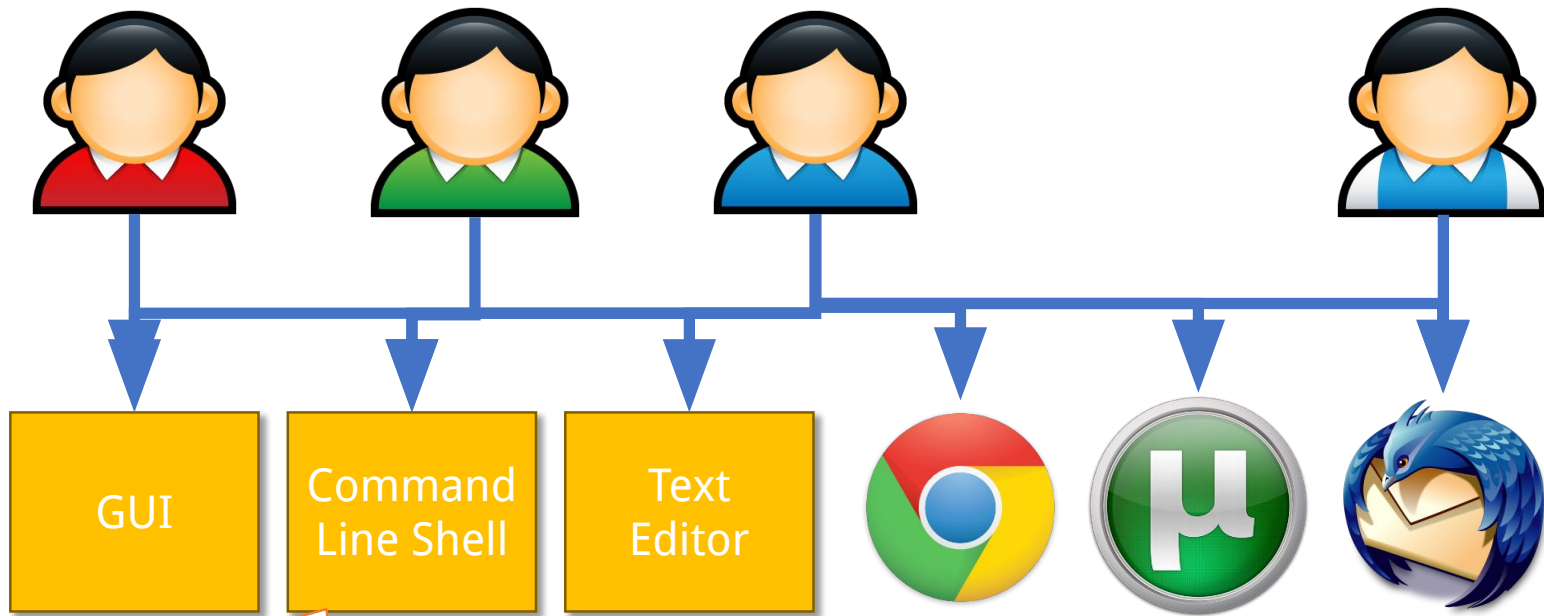
iOS

What is an Operating System?

- OS is software that sits between user programs and hardware



- OS provides interfaces to computer hardware
 - User programs do not have to worry about details



Shortly you will
be working in
the shell for
your lab and
homework!

Operating System

Hardware (e.g., mouse, keyboard)

htop

System statistics: Tasks: 295, 2689 thr: 1 running, 22.70/30.90 Load average: 0.50 0.81 1.91, Uptime: 1 day, 01:56:38

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME	Command
1	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
2	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
3	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
4	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
5	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
6	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
7	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
8	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
9	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
10	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
11	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
12	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
13	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
14	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
15	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
16	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
17	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
18	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
19	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
20	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
21	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
22	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
23	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
24	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
25	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
26	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
27	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
28	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
29	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
30	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
31	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
32	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
33	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
34	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
35	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
36	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
37	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
38	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
39	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
40	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
41	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
42	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
43	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
44	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
45	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
46	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
47	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
48	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
49	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
50	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
51	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
52	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
53	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
54	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
55	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
56	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
57	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
58	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
59	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
60	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
61	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
62	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
63	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
64	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
65	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
66	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
67	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
68	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
69	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
70	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
71	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
72	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
73	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
74	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
75	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
76	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
77	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
78	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
79	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
80	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
81	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
82	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
83	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
84	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
85	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
86	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
87	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
88	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
89	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
90	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
91	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
92	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
93	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
94	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
95	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
96	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
97	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
98	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
99	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop
100	root	0	0	1114K	1114K	0	S	0.0	0.0	0:01.13	htop

- OS is a resource manager and control program
 - Controls execution of user programs
 - Decides between conflicting requests for hardware access
 - Attempts to be efficient and fair
 - Prevents errors and improper use

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)

In this course, we will work in a POSIX Environment.

Who, what, why, Linux?

- 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) open

> **professional like gnu)**

> since april, and is sta

> things people like/disl

> (same physical layout o

> among other things).

>

> I've currently ported bas

> This implies that I'll get som

> I'd like to know what featur

> **are welcome, but I won't p**

>

> Linus (torv...@na.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 :-).**

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

```
→ khoury-cs3650.github.io git:(main) git pull
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 6 (delta 4), reused 6 (delta 4), pack-reused 0 (from 0)
Unpacking objects: 100% (6/6), 229.30 KiB | 990.00 KiB/s, done.
From github.com:Khoury-CS3650/khoury-cs3650.github.io
   3dad175..a343806  main      -> origin/main
Updating 3dad175..a343806
Fast-forward
 general.html | 12 ++++++++
img/ziming.jpg | Bin 0 -> 237621 bytes
 syllabus.html | 4 ++-
3 files changed, 14 insertions(+), 2 deletions(-)
create mode 100644 img/ziming.jpg
→ khoury-cs3650.github.io git:(main) git pull
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 7 (delta 3), reused 7 (delta 3), pack-reused 0 (from 0)
Unpacking objects: 100% (7/7), 610 bytes | 61.00 KiB/s, done.
From github.com:Khoury-CS3650/khoury-cs3650.github.io
   a343806..5bf6678  main      -> origin/main
Updating a343806..5bf6678
Fast-forward
 l/01/ferd/notes.html | 4 ++-
 syllabus.html        | 4 ++-
2 files changed, 4 insertions(+), 4 deletions(-)
```

Shell demo

- ls
- cd (cd ~, /, ..) : shell built-in command
- pwd : shell built-in command
- tree
- tab
- up/down arrow
- History
- htop

Why the command line?

- You might argue “I love GUI interfaces, so simple and sleek looking”
- 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

```
→ example-shell-script cat example.sh  
echo "Hello $1 $2"  
echo "What is your age?"  
read myAge  
echo "That is great you are $myAge years old!"
```

Example shell script Executing

```
→ example-shell-script ./example.sh Ziming Zhao  
Hello Ziming Zhao  
What is your age?  
500  
That is great you are 500 years old!
```

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)

```
LS(1) User Commands
NAME
  ls - list directory contents
SYNOPSIS
  ls [OPTION]... [FILE]...
DESCRIPTION
  List information about the FILES (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
  Mandatory arguments to long options are mandatory for short options too.
  -a, --all
    do not ignore entries starting with .
  -A, --almost-all
    do not list implied . and ..
  --author
    with -l, print the author of each file
  -b, --escape
    print C-style escapes for nongraphic characters
  --block-size=SIZE
    with -l, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below
  -B, --ignore-backups
    do not list implied entries ending with ~
  -c      with -lt: sort by, and show, ctime (time of last modification of file status information); with -l: show ctime and sort by name; otherwise
  -C      list entries by columns
  --color[=WHEN]
    colorize the output; WHEN can be 'always' (default if omitted), 'auto', or 'never'; more info below
  -d, --directory
    list directories themselves, not their contents
  -D, --dired
    generate output designed for Emacs' dired mode
```

Linux man pages are organized into numbered sections

Section 1: User Commands

- Executable programs and shell commands that regular users can run. Examples: `ls`, `cp`, `grep`, `gcc`

Section 2: System Calls

- Functions provided directly by the Linux kernel. Examples: `open()`, `read()`, `write()`, `fork()`

Section 3: Library Functions

- Functions provided by programming libraries (especially C library). Examples: `printf()`, `malloc()`, `strlen()`

Section 4: Special Files

- Device files and special files (usually in `/dev`). Examples: `/dev/null`, `/dev/random`

Section 5: File Formats

- Configuration file formats and conventions. Examples: `passwd` (for `/etc/passwd`), `fstab`, `hosts`

Section 6: Games

- Games and entertainment programs

Section 7: Miscellaneous

- Conventions, macro packages, and miscellaneous topics
- Examples: `ascii`, `regex`, `signal`

Section 8: System Administration

- Commands typically used by system administrators
- Examples: `mount`, `iptables`, `crontab`

Section 9: Kernel Routines

- Linux kernel API documentation (less commonly used)

Usage Examples:

- `man ls` - shows section 1 (user command)
- `man 2 open` - specifically shows section 2 (system call)
- `man 3 printf` - shows section 3 (library function)
- `man 4 tty`

You can see which sections contain a particular topic using `man -k keyword` or `apropos keyword`.

Xv6: A teaching operating system!

- <https://github.com/mit-pdos/xv6-public>

Xv6, a simple Unix-like teaching operating system

The latest version of xv6 is at: [xv6](#)

Introduction

Xv6 is a teaching operating system developed in the summer of 2006 for MIT's operating systems course, [6.828: Operating System Engineering](#).

History and Background

For many years, MIT had no operating systems course. In the fall of 2002, one was created to teach operating systems engineering. In the course students to multiple systems—V6 and Jos—helped develop a sense of the spectrum of operating system designs.

V6 presented pedagogic challenges from the start. Students doubted the relevance of an obsolete 30-year-old operating system written in an obscure language. In 2006, we had decided to replace V6 with a new operating system, xv6, modeled on V6 but written in ANSI C and running on multiprocessor Intel processors (instead of using special-case solutions for uniprocessors such as enabling/disabling interrupts) and helps relevance. Finally, writing a new

xv6

- We will be using xv6 to build and implement some Operating Systems features
- This will give you experience adding features to a large piece of software.