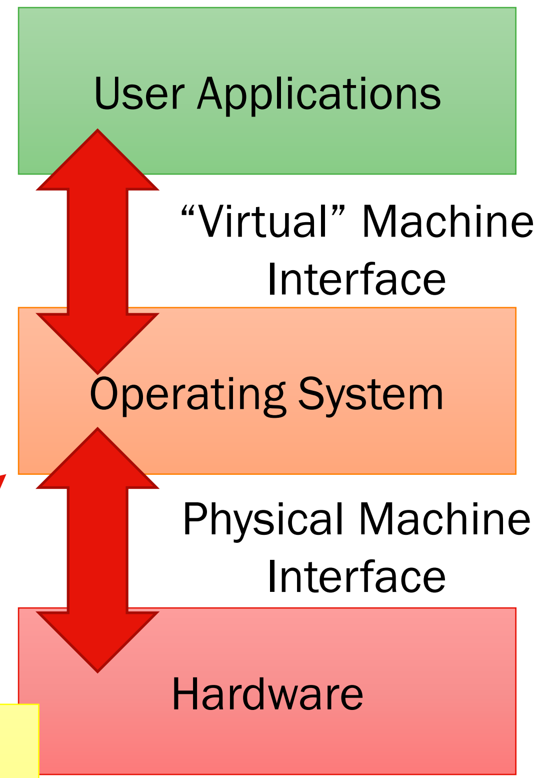
CS5460: Operating Systems

Lecture 1: Course Overview

(Chapter 2)

What is an Operating System?

- Interface between user and hardware
  - Exports simpler “virtual machine” interface
  - Hides ugly details of real hardware
- Manages shared resources
  - CPU, memory, I/O devices, …
  - Ensure fairness
  - Protects processes from one another
- Provides common services
  - File system, virtual memory, network, CPU scheduling, …
- Goals:
  - Convenient to use
  - Efficient



OS versus Kernel?
Kernel is the core and most privileged part of an OS

OS Techniques for Easier Apps

- Abstraction
  - Often over physical resources: cores, memory, disk, net (virtualization)
  - Sometimes others: pipes, process groups, ulimits, futexes
- Advantages?
  - Portability
    - Applications run on more diverse hardware
  - Higher-level
    - Easier to express complex applications
    - May be able to accelerate implementations without changing applications
- Challenges
  - What are the right abstractions? (often virtualizations)
  - What is the cost of abstraction?

Make sure you can name the abstractions for each of these resources

OS Techniques for Easier Apps

- Resource management and sharing
  - Inter application protection and isolation
  - Easy allocation and sharing of resources
  - Fair access to resources
- Challenges
  - What are the right **mechanisms**?
    - For safe isolation?
    - For sharing resources?
  - What are the right **policies**?
    - Efficiency, fairness?

What is the difference?

What are examples?

What is the benefit of separating them?

Our route: Three Easy Pieces

- Virtualization
  - Processes, system calls, context switch, scheduling
  - Address spaces, virtual addressing, virtual memory
- Concurrency
  - Threads, locking primitives
- Persistence
  - I/O Devices, Disks, Flash, File systems
  - NFS and a bit of networking and distributed systems

Each of these topics first arose in the context of operating systems

Virtualizing Cores

```
$ numactl -C 1 /bin/bash
$ while true; do sleep 1; echo A; done & \
  while true; do sleep 1; echo B; done &
[1] 27883
[2] 27884
A
B
A
B
A
B
```

Virtualizing Memory

```
void main() {
    int* p = malloc(sizeof(int));
    while (1) {
        sleep(1);
        printf("%p = %d\n", p, *p);
        (*p)++;
    }
}

$ ./mem & ./mem &
[1] 28481
[2] 28482
0x555555559260 = 0
0x555555559260 = 0
0x555555559260 = 1
0x555555559260 = 1
```

Persistence

```
void main() {
    int fd = open("foo",
                  O_RDWR | O_CREAT | O_TRUNC,
                  S_IRWXU);
    assert(fd != -1);
    int rc = write(fd, "test\n", 5);
    assert(rc == 5);
    close(fd);
}
```

```
$ ./io
$ cat foo
test
```

No mention of devices, disks, drivers, hardware.

Where/how are I/O devices accessed here?

What keeps programs from e.g. skipping the filesystem code that does permissions checks?

Managing Concurrency

```
int i = 0;

void* run(void* _) {
    for (int j = 0; j < 1000000; j++) i++;
}

void main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, run, NULL);
    pthread_create(&t2, NULL, run, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("%d\n", i);
}
```

```
$ ./inc
1041048
$ ./inc
1087180
```

☠️
Please don't write code like this

Why Take OS?

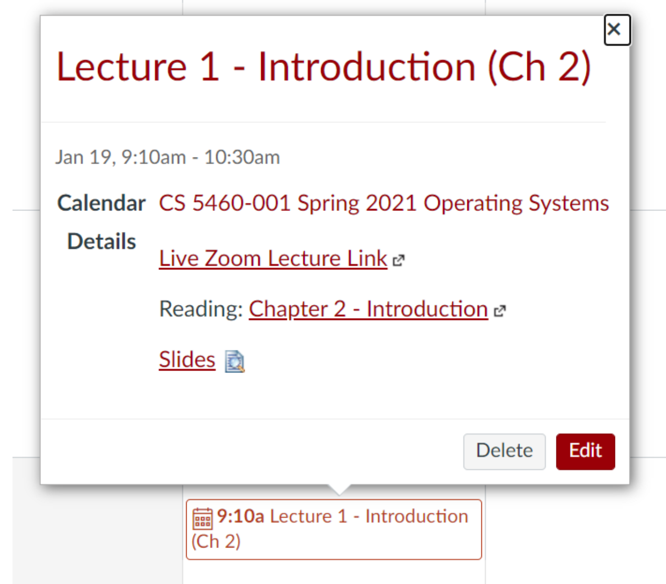
- Remove doubt about how programs work
  - We take abstractions for granted, but they have implementations
    - Threads, processes, libraries, files, sockets
  - You should understand what happens when
    - A program is run
    - A process accesses a file
    - A process uses more memory than is physically available
    - A mouse is moved and the pointer moves on the screen

- Write better, more efficient code
  - Good code at one layer requires understanding how the layer below will carry things out
  - Learn performance engineering; OS is performance obsessed

- Learn interface design and tradeoffs by example

- Demystify kernel programming

This Course



- Instructor: Ryan Stutsman
- TAs: John, Amit, Vinita, and Calvin
- Textbook: Operating Systems: Three Easy Pieces v1.0
  - ostep.org
- Required background:
  - Undergrads: CS 4400 and ability to program in C
  - Grads: Ability to program in C
- Office hours on calendar
  - (Instructor Tue 2-3:30 & Thu 10:45-12:15)

Canvas

- Canvas will be used for announcements
  - Make sure you are setup to get notified of announcements via email
- Primarily **use Canvas Discussions** for questions
  - In general, try to keep discussions open
  - Send private messages when necessary
- Canvas Inbox if you need to contact us directly

Office Hours & MS Teams

- Offices hours are listed near top of syllabus
- MS Teams "Office Hours" channel
  - TAs will create a thread when they are available
  - Message them there or directly during their hours
  - They can chat, screen share, etc.
- Will also use this answer questions during exam
- Use Canvas Discussions for most general Q&A

Lectures

- Live lecture via Zoom
  - Ask questions via Zoom chat
  - I won't be able to answer all questions; will do my best
  - I will have my camera on
  - Your camera is up to you; if you have it on be mindful
  - You can unmute, but respect the question flow in chat
- Lectures will be recorded
  - Posted with <= 24 hours delay on Canvas
- **Attend, participate, and pay attention**
  - Only required attendance: Midterm (3/16) & Final (5/5)

Assignments

- 2 to 4 Homeworks (10%)
  - Shorter and more focused on reinforcing concepts
- 4 or 5 Assignment (40%)
  - You'll write C code
  - **Some of them are time consuming – start early**
  - Graded on CADE lab Linux machines; get account if needed
    http://www.cade.utah.edu/
- Exams (50%): Tie together concepts from lectures and projects
  - Students do well on projects; places extra importance on exams
  - Covers OSTEP readings, homework and projects, and lecture content
  - Open-book on Canvas during class/final exam time (3/16, 5/5)

Grading & Late Policy

- Standard 90/80/70/60 grading scale
- Late turn in excused for family/medical emergency
  - Please make arrangements in advance, if possible
- Each student has five late days to use on programming assignments only (Assignment 1-5)
  - Notify Lead TA for assignment how many late days you want to use for that assignment
- Late assignments incur 10% penalty per day (after late days have been applied)
  - <= 1 day late, 90% credit; <= 2 days, 80%; <= 3 days, 70%
  - > three days late receives 60% credit

Collaboration vs. Cheating

- To participate in this course you must:
  - Read SoC Policy on Academic Misconduct (see syllabus)
  - **Have an Acknowledgement Form filed in SoC office by end of second week otherwise you must withdraw or receive an EU**
- Do not…
  - Copy code from another student
  - Even look at code from another student
  - Copy code from the web
  - Ask for answers on StackOverflow or a similar web site
- It is okay to discuss solution strategies with classmates
- All forms of cheating/misconduct result in course failure
- We will check for cheating

Start early on assignments!

Classroom Expectations

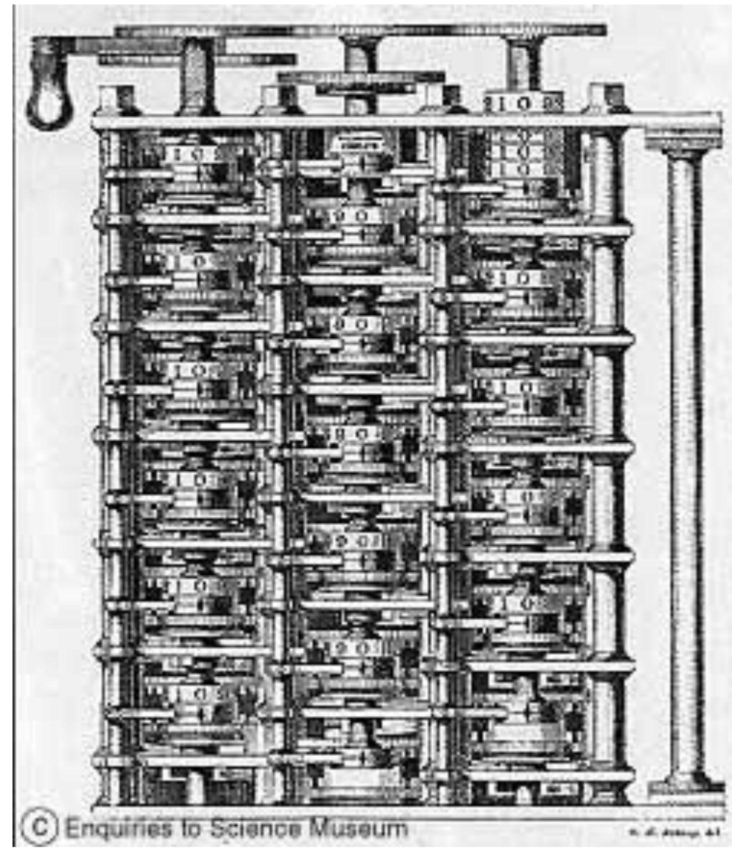
- Zoom, Canvas, and Teams are our classroom
  - Pay one another respect you would in any classroom
- Examples of potential harassment/misconduct
  - Sending unwanted and unsolicited messages
  - "Pinning" video of students other than instructor or speaker
  - Unflattering, intimidating, or demeaning statements about anyone especially remarks related to race, color, origin, religion, sex, gender identity/expression, sexual orientation, background, veteran status, educational status, genetics

Concerns? Tell me if you feel comfortable doing so
Else: dept. advisors or Office of Equal Opportunity

A Brief History of Operating Systems

Prehistory (pre-1945)

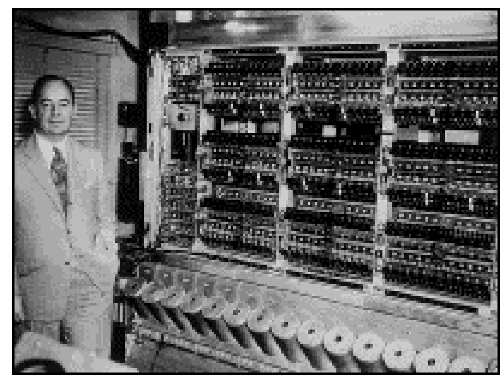
- Charles Babbage (1792-1871) & Ada, Countess of Lovelace (1815-1852)
    - 1st computer architect/programmers
    - First digital computer
    - "Analytical engine"
    - Never actually got it to work (although others subsequently have)
    - No operating system: programmer programmed to raw hardware



Babbage Analytical Engine

1930s, 1940s

- Slow human "computers" → faster machines
- Mechanical relays, vacuum tubes, plug-boards, core memory:
  - Turing (the "Bombe")
  - Aiken (Harvard architecture)
  - Von Neumann (Princeton IAS)
  - Eckert and Mauchley (ENIAC)
  - Zuse (Z1, Z3)
- Huge, hot, fragile, and slow by modern standards



Von Neumann

OS History: Phase I

Hardware is <u>very</u> expensive, humans are cheap!

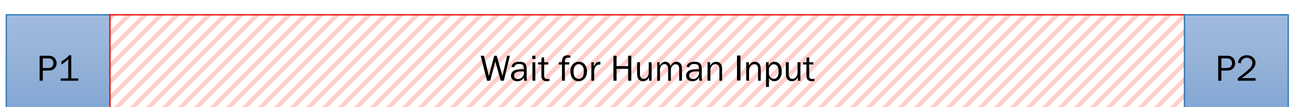
- One user at the console
  - One function at a time (no overlap between computation and IO)
  - User sitting at console to debug
  - First OSes: Common library routines

IBM Model 701 (Early 1950's)

Rent $20,000 a month *in 1953*
~$180k/mo today

16 KHz, 2048 words of 36 bits, ~75 pixels

Uniprogrammed

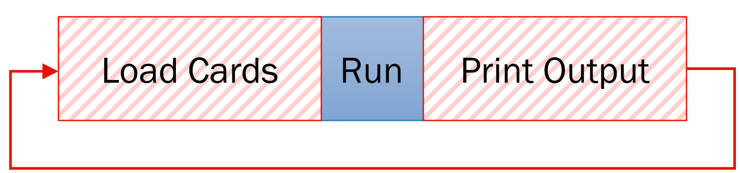


OS History: Phase I

- Batch processing: load, run, print, dump, repeat
  - Put up glass walls around computer
  - Users give program (cards or tape) to human who schedules jobs
  - OS loads, runs, and dumps user jobs
  - Non-interactive batch processing
  - Efficient use of HW, at least while not loading decks
  - Debugging hard, core dumps
  - Short jobs starve behind large ones
- No overlap of I/O and compute

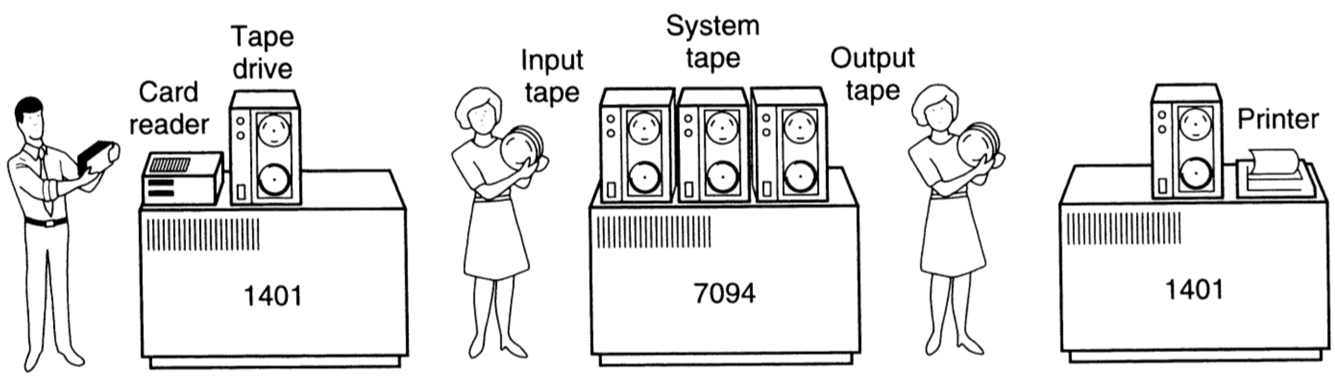
Lost interactive debugging → Humans spend more time waiting

Uniprogrammed

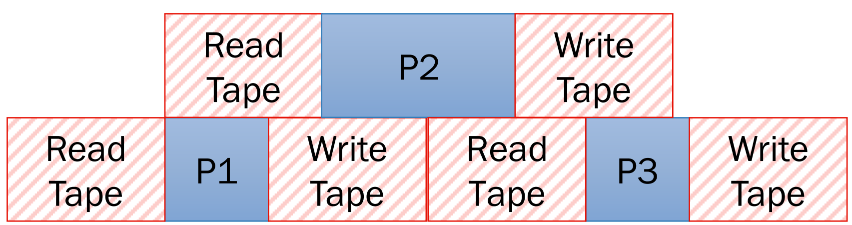


OS History: Phase I

- Data channels and interrupts (early ‘DMA’)
  - Buffering and interrupt handling in OS (“batch monitor”)
  - No protection – one job running at a time!
  - Overlaps running computation and IO in parallel
    - “Spooling”
  - Users carried around permanent storage (cards, tapes, …)



Uniprogrammed but overlapped I/O

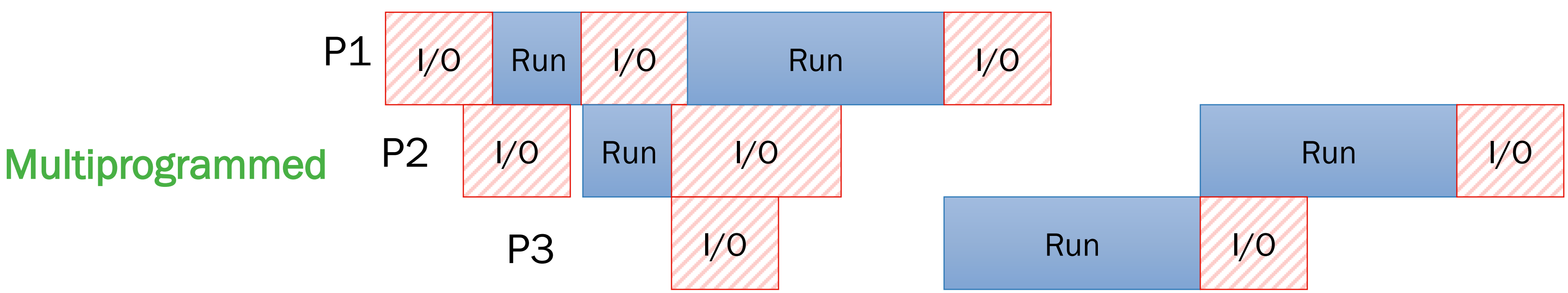


IBM 7094 (Early 1960's)

Just
$400k/mo!

OS History: Phase I

- **Multiprogramming**; memory protection
  - More memory – can load several jobs at once
  - OS (monitor) always resident to coordinate activities
  - OS manages interactions between concurrent jobs:
    - Runs programs until they block due to I/O
    - Decides which blocked jobs to resume when CPU freed
    - Protects each job's memory from other jobs
  - IBM/360 combined IBM 1401 and IBM 7094
    - First machine to use ICs instead of individual transistors
- OS disasters → software engineering



OS History: Phase II (1965-1980)

Hardware is cheap, humans are expensive!

- Timesharing
  - First time-share system: CTSS from MIT (1962)
  - Timer interrupts: enable OS to take control (pre-emptive multitasking)
  - MIT/Bell Labs/GE collaboration led to MULTICS
    - Envisioned one huge machine for all of Boston (!!!)
    - Started in 1963, “done” in 1969, dead shortly thereafter
    - Bell Labs bailed on project, GE bailed on computers!
  - DEC PDP minicomputers: start of bottom feeding frenzy
    - PDP-1 in 1961 (4K 18-bit words, $120,000)
    - Thompson and Ritchie
    - “C” language developed for Unix
    - Guiding principle of UNIX: Keep it simple so it can be built

OS History: Phase II (1965-1980)

- Timesharing (continued)
  - **Terminals are cheap**
    - Let all users interact with the system at once
    - **Debugging gets a lot easier**
    - Process switching occurs much more frequently
  - New OS services:
    - Shell to accept interactive commands, debuggers
    - File system to store data persistently
    - Virtual memory to allow multiple programs to be resident
  - New problems: response time and thrashing
    - Need to limit number of simultaneous processes or you can fall off performance cliff (“login”)
    - Lots of work on resource scheduling (CPU, memory, I/O)

OS History: Phase III (1980-2000s)

- Personal computing: every “terminal” has computer
  - One user per machine (remind you of anything?)
  - Initial PC OSes similar to old batch systems (w/TSR hacks)
  - Advanced OS features crept back in!
    - Linux, macOS, and Windows (starting with NT) now all include the ideas pioneered in the earlier decades

Original IBM PC

A young Bill Gates

OS History: Phase IV (2000s—now)

- Lots and lots of computers per person
  - Embedded systems
    - Cars commonly have 50+ processors
    - Cars, airplanes, factories run a huge amount of software
  - Mobile computing
    - PCs exceed the needs of many current computer users
    - Rise of smart phones and tablets
  - Cloud computing and serverless
    - Virtualized compute resources flexibly allocated on demand
    - Computing as a service rather than devices

The Future of OSes

	1983	2016	
MIPS	0.5	200,000+	400,000x
$/MIP	$100,000	$0.005	20,000,000x
Memory	1 MB	3+ TB	3,000,000x
Network	0.1 Mbps	100,000 Mbps	1,000,000x
Storage	20 MB	10s of TB	500,000x
Address size	16 bits	64 bits	

- The very small
- The very large
- Change
  - 1953-2003: 10 orders of magnitude
  - 1983-2016: →
  - Nothing like it in other fields:
    - Transportation: 100x
    - Communication: 10^7

Characteristics of Modern OSes

- Enormous
  - Millions of lines of code, 1000s of engineer years
- Complex
  - Asynchronous, hardware idiosyncrasies, performance hacks
- Poorly understood
  - Systems outlive their builders
  - Never debugged: OS/360 released with 1000 bugs
  - Behavior hard to predict, security flaws
  - Unreliable
- Incredibly successful
  - Hard to imagine modern programming without them

Next time: Processes

- OSTEP Chapters 4, 5, 6

Important Terms and Ideas

- Operating System and Kernel
- Abstraction and Virtualization
- Policy and Mechanism
- Uniprogramming and Multiprogramming