CS 5600 Computer Systems

Spring 2026

Introductory Material
Lecture 0
January 13, 2026

Prof. Scott Valcourt

s.valcourt@northeastern.edu

603-380-2860 (cell)

Portland, ME

The Roux Institute
Northeastern University

Questions answered in this lecture:

- Who is in this course?
- What will you do in this course?
- What is an OS and why do you want one?
- Why study operating systems?

- Actions for you:
  - Review the course Canvas pages and first programming project
  - Prepare your working environment

Hello!

- Welcome to CS 5600
  - Are you in the right classroom?
- Who am I?

Scott Valcourt

- Roux Institute, Portland, ME
- s.valcourt@northeastern.edu

- PhD Engineering: Systems Design & Cognate in College Teaching
- MS: Computer Science
- BA: Computer Science with Mathematics Emphasis

- Cyberinfrastructure, Broadband Communications, Systems, Telehealth, CS Education

Scott Valcourt

- Served as the second director of the UNH InterOperability Lab (IOL)
  - I killed most of the dead networking technologies
- Served as the PI on NH's NTIA BTOP ARRA-funded network construction project that created NetworkNH
- Co-Founder of the UNH Telehealth Practice Center

- I have been a Scout for over 47 years!
- I am a native Mainer!
- We have a vegetable farm.

University of New Hampshire InterOperability Laboratory

NetworkNH Broadband Services for New Hampshire

Telehealth Practice Center

Getting to Know You

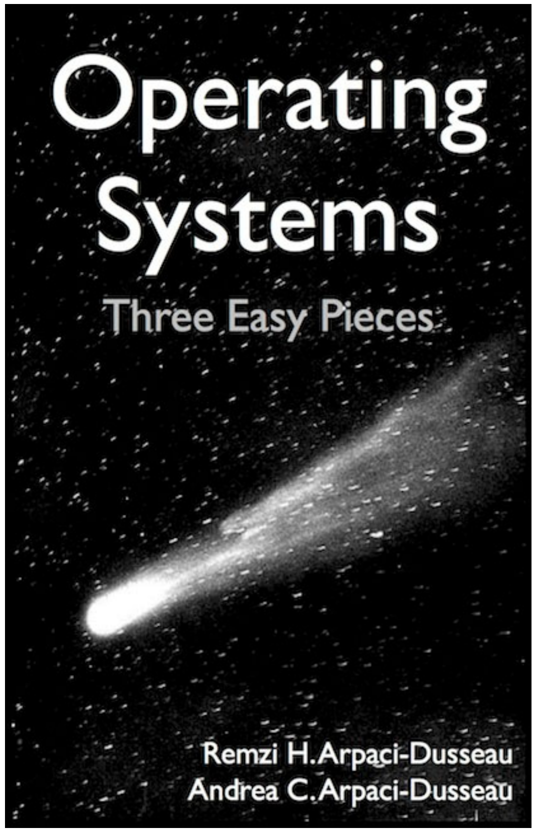
- Pair with Another Student
- Exchange the following information about each other:
  - Their name (and preferred name, if any)
  - Their pronouns
  - Their hometown
  - Their undergraduate degree and from which institution
  - Their favorite activity to enjoy if not in class right now
- Circulate through the room and Introduce your partner

Class Norms

- Interaction
  - Chat/Ask a question anytime
  - Raise Hand to ask a question and open microphone when called
  - Attempt to follow a class discussion back-off algorithm
- Process
  - Wednesday Class Cadence
- Class Recording
  - Can we agree to allow for the recording of the class, removing video, yet maintaining audio questions and discussion?

Textbook

- Operating Systems: Three Easy Pieces
  - Remzi and Andrea Arpaci-Dusseau
- Free, PDFs available online at http://pages.cs.wisc.edu/~remzi/OSTEP/



Documents to Download

- Thompson and Ritchie original UNIX paper – Homework Reading for Next Week

- Helpful Tutorial Reference Document

Resources

- Khoury Account
  - https://www.khoury.northeastern.edu/life-at-khoury/systems/getting-started/
  - my.khoury.northeastern.edu
- Khoury Systems Knowledge Base
  - https://www.khoury.northeastern.edu/life-at-khoury/systems/knowledge-base/
- Khoury GitHub
  - https://northeastern.service-now.com/kb_view.do?sysparm_article=KB0012152
- Putty
  - https://www.chiark.greenend.org.uk/~sgtatham/putty/
- ssh login.khoury.northeastern.edu

Reaching Out

- Office Hours
  - Prof. Valcourt:
    - Wednesdays 11:00am-12:00pm ET
    - By Appointment

- Email
  - Prof. Valcourt: s.valcourt@northeastern.edu

- Phone Call
  - 603-380-2860 (text or cell call)

Say Hi to the TA

- Krunal Ghanshyambhai Savaj
- savaj.k@northeastern.edu
- Office Hours
  - Coming Soon
- https://northeastern.zoom.us/j/

Why Take This Course?

- Computers are everywhere
  - In your pocket
  - In your microwave
  - Up in space
- We take hardware and OS features for granted
  - Double click and your program loads
  - Devices just work (most of the time…)
  - Buggy apps can't crash your machine
- … but very few people truly understand how computers really work, at a low-level

Goals

- Fundamental understanding about computer hardware and operating systems
  - From the moment a PC boots up
  - … to managing devices and memory…
  - … up to loading complex, threaded applications
- Focus on software and systems
  - Not hardware
  - No theory
- Project-centric, hands-on experience
  - You will build OS components in this class
  - This will be a **huge** amount of work
  - But you will also learn a **huge** amount

At the end of this course…

- You will understand low-level details of computer hardware and modern CPUs

- You will know the key functions of Operating Systems
  - Managing I/O devices and memory
  - Loading programs
  - Scheduling the CPU and isolating processes

- You will understand that designing systems is an art, not a science
  - Building systems is about managing tradeoffs

Online Resources

- Canvas CS5600 Course Page(s)
- Class forum is on Piazza
  - Sign up today!
  - Install their iPhone/Android app (if you wish)
- When in doubt, post to Piazza
  - Piazza is preferable to email
  - Use #hashtags (#lecture2, #project3, etc.)

Today	PC Hardware, CPUs, and OS Basics
2	Scheduling and Switching
3	Processes and Threads
4	Address Translation and Virtual Memory
5	Synchronization and Deadlock
6	Parallel Programming
7	Storage, Disks, and SSDs
8	RAID
March 3-7	**Spring Break**
9	Files and Directories
10	File Systems
11	Memory Management; Garbage Collection; Security
12	Virtual Machine Monitors
13	Authorization, Access Control, and Exploit Prevention
14	Networking
April 24	**Final Exam** (Apr 22 – Apr 24)

Teaching Style

- 3-hour lectures
  - Breaks every hour. Other suggestions?
- I have been working with systems for a long time
  - Things make sense to me may not make sense to you
  - I talk fast if nobody stops me
- Solution: ask questions!
  - Seriously, ask questions
  - Standing up here in silence is very awkward
  - I will stand here until you answer my question

Workload

Homework (10)	1% each for 10% total
Quizzes (10)	1% each for 10% total
Programming Assignments (4)	10% each for 40% total
Course Project	20%
Final Exam	20%

Projects

- This course is project-centric
  - You will be building large portions of an operating system
  - Start early!
  - Seriously, start early!
- 4 projects/programming assignments
  - Due at 11:59:59pm on specified date
  - Usually providing 3+ weeks to complete the work
  - Use GitHub to submit your code, documentation, etc.
  - **Working code is paramount**

Project Groups

- Programming Projects will be completed in groups of three
- You may choose your own partners
  - You may switch partners between projects
  - Do not complain to me about your lazy partner
    - Hey, you picked them
- Can’t find a partner?
  - Post a message on Piazza!
- Want to work alone?
  - You can, but you become a team of one

Late Policy for Projects

- Each student is given 8 *time travel days*
  - May be used to extend project deadlines
    - Example: 1 project extended by 4 days
    - Example: 2 projects each extended by 2 days
  - **You don't need to ask me**, just turn-in stuff later than the due date
  - All group members must have unused *time travel days*
    - i.e., if one member has zero *time travel days* left, the whole group is late
- Assignments are due at 11:59:59 PM, **no exceptions**
  - 20% off per day late
  - 1 second late = 1 hour late = 1 day late

Participation

- This is a masters level course
  - I’m not taking attendance
  - I don’t care if you skip lecture
- However, participation is noticed and matters
  - Be active on Piazza
  - Ask questions in lecture
  - Answer questions that I ask in lecture
- Ideally, I want to know everyone’s name by the end of the semester

Exam

- Final
  - Approximately 3 hours
  - The final will be **cumulative**
- The exam is:
  - Open book
  - Open notes
  - Open computer
  - Take-Home

Learning Approach

- Each student is given 2 *attempts* to complete programming assignments, homeworks, and quizzes (not exams)
  - *If you did not achieve a mastery score (85+),* you can resubmit that work for one additional regrade
- If you think there has been a grading error, come to my office hours
- **When you have used both *attempts*, you cannot ask for regrades and the best of the two scores will stand. Therefore, be focused and precise.**

Cheating

- **Do not do it**
  - Seriously, don’t make me say it again
- Cheating is an automatic zero
  - Will be referred to the university for discipline and possible expulsion
- For projects: code must be original, written by you and your groupmates only
  - Starter code obviously doesn’t count
  - StackOverflow/Quora/GitHub/ChatGPT are <u>not</u> your friends
  - If you have questions about an online resource, ask us

Final Grades

- At the end of the semester, all your grades will sum to 100 points

$$\overbrace{10 + 10 + 10 + 10}^{\text{PAs}} + \overbrace{20 + 20}^{\text{Proj \& Exam}} + \overbrace{10 + 10}^{\text{Quizzes \& HW}} = 100$$

- Final grades are based on a simple scale:
  - A >92, A- 90-92, B+ 87-89, B 83-86, B- 80-82, …
- I don’t curve grades
- All grades are rounded up

Expectations

- You have had at least one programming course (probably Python), and have had experience with C programming
- You can write a program on your own and can also program as part of a team (at least with one other programmer)
- You can read/skim technical material that is beyond your current expertise – reading in detail until you get stuck, then skimming over the detailed stuff
- We are going to learn/use the C programming language in this course
  - It feels a lot like Java – it is one of the ancestors of Java
  - If you are new to C, there are many tutorials to get you up to speed

Final Planning Thoughts

- Any other norms or procedures we want to identify and agree to in our learning environment?

- Partway through the semester (around the MidTerm Exam), I will ask if there are any topics that you want to include in the course – we have one spare week in my topics list

- Any questions?

CS 5008 Data Structures, Algorithms, and Apps

Spring 2026

Overview of Computer Systems
Lecture 1
January 13, 2026

Prof. Scott Valcourt

s.valcourt@northeastern.edu

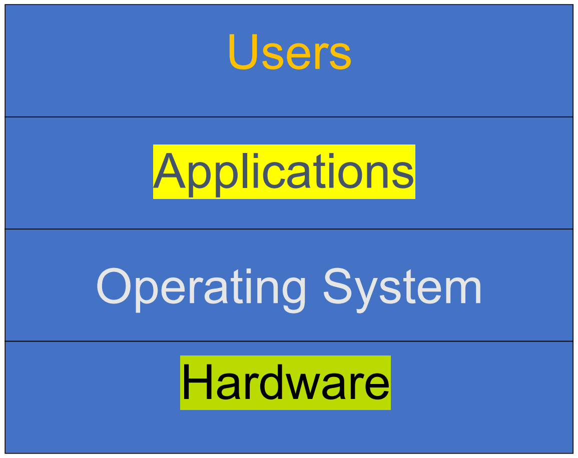
603-380-2860 (cell)

Portland, ME

The Roux Institute
Northeastern University

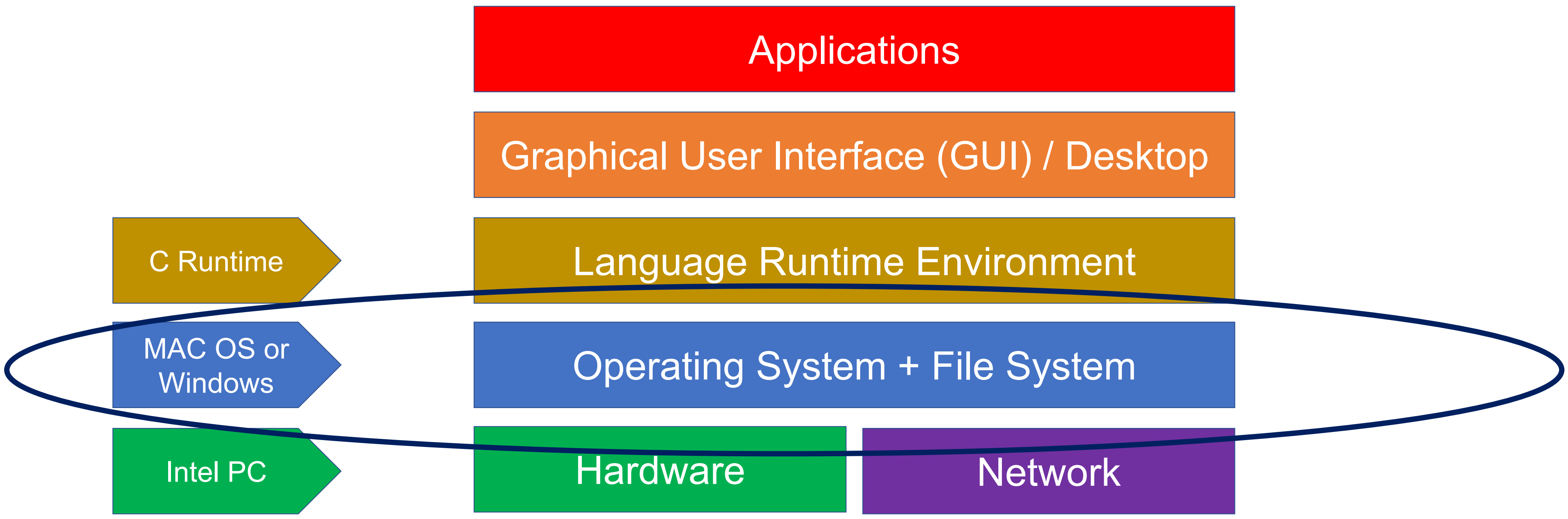
What is an Operating System?

Not easy to define precisely…



Operating System (OS):
Software that converts hardware into a useful form for applications

A very simplified view of a computer system


Applications
Graphical User Interface (GUI) / Desktop
C Runtime
Language Runtime Environment
MAC OS or Windows
Operating System + File System
Intel PC
Hardware
Network

What DOES the OS Provide?

Role #1: Abstraction - Provide standard library for resources

What is a resource?
- Anything valuable (e.g., CPU, memory, disk)

What abstraction does modern OS typically provide for each resource?
- CPU:
  - process and/or thread
- Memory:
  - address space
- Disk:
  - files

Advantages of OS providing abstraction?
- Allow applications to reuse common facilities
- Make different devices look the same
- Provide higher-level or more useful functionality

Challenges
- What are the correct abstractions?
- How much of hardware should be exposed?

What DOES the OS Provide?

Role #2: Resource management – Share resources well

Advantages of OS providing resource management?
- Protect applications from one another
- Provide efficient access to resources (cost, time, energy)
- Provide fair access to resources

Challenges
- What are the correct mechanisms?
- What are the correct policies?

OS Organization

- How to cover all the topics relevant to operating systems?

Three PIECES: FIRST

- Virtualization
  - Make each application believe it has each resource to itself
- Demo
  - Virtualize CPU and memory

Three PIECES: SECOND

- Concurrency: Events are occurring simultaneously and may interact with one another

- OS must be able to handle concurrent events

- Easier case
  - Hide concurrency from independent processes

- Trickier case
  - Manage concurrency with interacting processes
    - Provide abstractions (locks, semaphores, condition variables, shared memory, critical sections) to processes
    - Ensure processes do not deadlock

- Demo
  - Interacting threads must coordinate access to shared data

Three PIECES: THIRD

- Persistence: Access information permanently
  - Lifetime of information is longer than lifetime of any one process
  - Machine may be rebooted, machine may lose power or crash unexpectedly

- Issues:
  - Provide abstraction so applications do not know how data is stored : Files, directories (folders), links
  - Correctness with unexpected failures
  - Performance: disks are very slow; many optimizations needed!

- Demo
  - File system does work to ensure data is updated correctly

Advanced Topics

- Current systems
  - Multiprocessors
  - Networked and distributed systems
  - Virtual machines

Why study Operating Systems?

- Build, modify, or administer an operating system

- Understand system performance
  - Behavior of OS impacts entire machine
  - Tune workload performance
  - Apply knowledge across many layers
    - Computer architecture, programming languages, data structures and algorithms, and performance modeling

- Fun and challenging to understand large, complex systems

What is an Operating System?

- When you boot up a machine, you see your operating system booting up
  - Mac
  - Windows
  - Linux

Many Different Operating Systems (OSs)

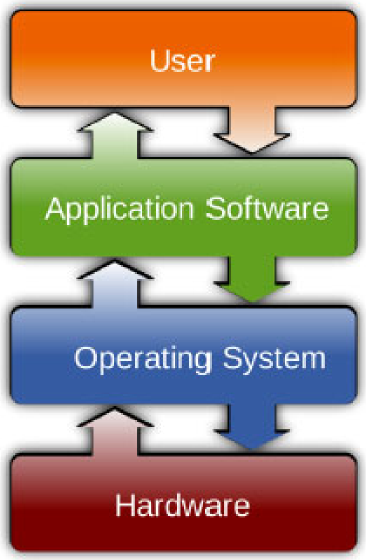
- Windows
- Linux
- BSD

What is an Operating System?

- OS is software that sits between an application and the hardware

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

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

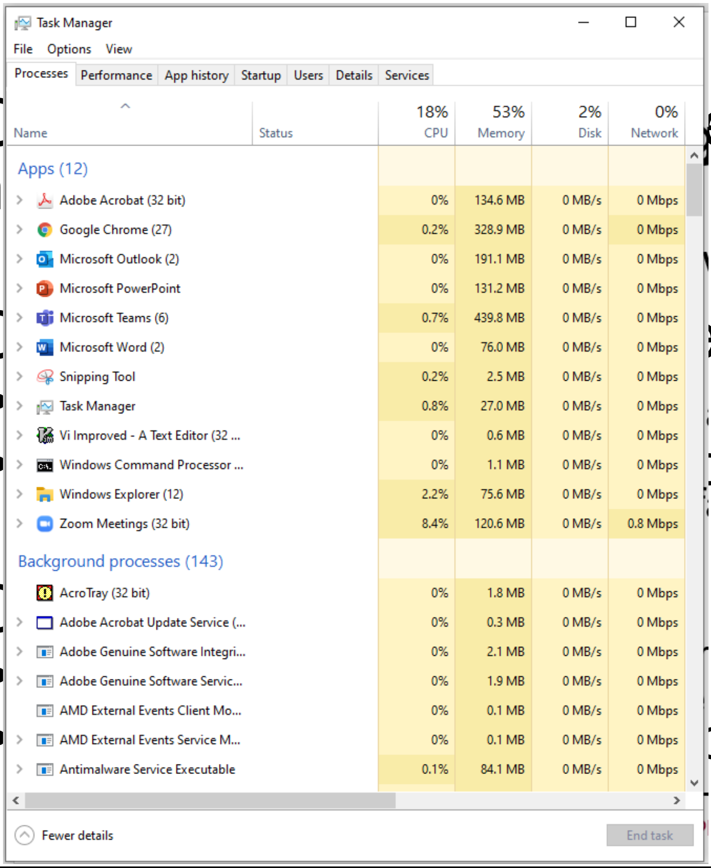


What is an Operating System?

•C etwee
h

•C er and
• g requ
• d fair i

•C
• programs
• per use



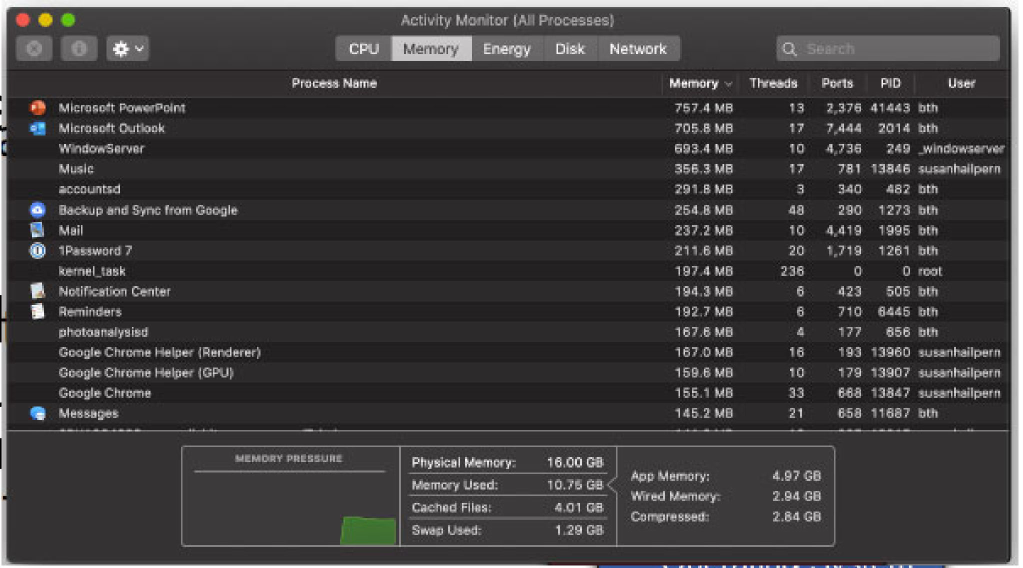
Task Manager

File Options View

Processes | Performance | App history | Startup | Users | Details | Services

Name	Status	18% CPU	53% Memory	2% Disk	0% Network
Apps (12)					
Adobe Acrobat (32 bit)		0%	134.6 MB	0 MB/s	0 Mbps
Google Chrome (27)		0.2%	328.9 MB	0 MB/s	0 Mbps
Microsoft Outlook (2)		0%	191.1 MB	0 MB/s	0 Mbps
Microsoft PowerPoint		0%	131.2 MB	0 MB/s	0 Mbps
Microsoft Teams (6)		0.7%	439.8 MB	0 MB/s	0 Mbps
Microsoft Word (2)		0%	76.0 MB	0 MB/s	0 Mbps
Snipping Tool		0.2%	2.5 MB	0 MB/s	0 Mbps
Task Manager		0.8%	27.0 MB	0 MB/s	0 Mbps
Vi Improved - A Text Editor (32 ...		0%	0.6 MB	0 MB/s	0 Mbps
Windows Command Processor ...		0%	1.1 MB	0 MB/s	0 Mbps
Windows Explorer (12)		2.2%	75.6 MB	0 MB/s	0 Mbps
Zoom Meetings (32 bit)		8.4%	120.6 MB	0 MB/s	0.8 Mbps
Background processes (143)					
AcroTray (32 bit)		0%	1.8 MB	0 MB/s	0 Mbps
Adobe Acrobat Update Service (...		0%	0.3 MB	0 MB/s	0 Mbps
Adobe Genuine Software Integri...		0%	2.1 MB	0 MB/s	0 Mbps
Adobe Genuine Software Servic...		0%	1.9 MB	0 MB/s	0 Mbps
AMD External Events Client Mo...		0%	0.1 MB	0 MB/s	0 Mbps
AMD External Events Service M...		0%	0.1 MB	0 MB/s	0 Mbps
Antimalware Service Executable		0.1%	84.1 MB	0 MB/s	0 Mbps

Fewer details | End task



Activity Monitor (All Processes)

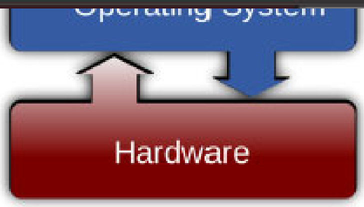
CPU | Memory | Energy | Disk | Network | Search

Process Name	Memory	Threads	Ports	PID	User
Microsoft PowerPoint	757.4 MB	13	2,376	41443	bth
Microsoft Outlook	705.8 MB	17	7,444	2014	bth
WindowServer	693.4 MB	10	4,736	249	_windowserver
Music	356.3 MB	17	781	13846	susanhailpern
accountsd	291.8 MB	3	340	482	bth
Backup and Sync from Google	254.8 MB	48	290	1273	bth
Mail	237.2 MB	10	4,419	1995	bth
1Password 7	211.6 MB	20	1,719	1261	bth
kernel_task	197.4 MB	236	0	0	root
Notification Center	194.3 MB	6	423	505	bth
Reminders	192.7 MB	6	710	6445	bth
photoanalysisd	167.6 MB	4	177	656	bth
Google Chrome Helper (Renderer)	167.0 MB	16	193	13960	susanhailpern
Google Chrome Helper (GPU)	159.6 MB	10	179	13907	susanhailpern
Google Chrome	155.1 MB	33	668	13847	susanhailpern
Messages	145.2 MB	21	658	11687	bth

MEMORY PRESSURE

Physical Memory:	16.00 GB
Memory Used:	10.75 GB
Cached Files:	4.01 GB
Swap Used:	1.29 GB

App Memory:	4.97 GB
Wired Memory:	2.94 GB
Compressed:	2.84 GB



Most Common Operating System 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 or handheld gaming device or a smartphone)

What a happens when a program runs?

•A running program executes instructions.

1. The processor **fetches** an instruction from memory.
2. **Decode**: Figure out which instruction this is
3. **Execute**: i.e., add two numbers, access memory, check a condition, jump to function, and so forth.
4. The processor moves on to the **next instruction** and so on.

Operating System (OS)

- Responsible for
  - Making it easy to **run** programs
  - Allowing programs to **share** memory
  - Enabling programs to **interact** with devices

OS is in charge of making sure the system operates correctly and efficiently.

Virtualization

- The OS takes a physical resource and transforms it into a virtual form of itself.
  - **Physical resource**: Processor, Memory, Disk …
- The virtual form is more general, powerful and easy-to-use.
- Sometimes, we refer to the OS as a **virtual machine**.

System Call

- System call allows user **to tell the OS what to do**.
  - The OS provides some interface (APIs, standard library).
  - A typical OS exports a few hundred system calls.
    - Run programs
    - Access memory
    - Access devices

The OS is a resource manager.

- The OS **manage resources** such as *CPU*, *memory* and *disk*.
- The OS allows
  - Many programs to run → Sharing the <u>CPU</u>
  - Many programs to *concurrently* access their own instructions and data → Sharing <u>memory</u>
  - Many programs to access devices → Sharing <u>disks</u>

Virtualizing the CPU

- The system has a very large number of virtual CPUs.
  - Turning a single CPU into a <u>seemingly infinite number</u> of CPUs.
  - Allowing many programs to <u>seemingly run at once</u>

    $\rightarrow$ **Virtualizing the CPU**

Virtualizing the CPU (Cont.)

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>
#include "common.h"

int
main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char *str = argv[1];
    while (1) {
        Spin(1); // Repeatedly checks the time and returns once it has run for a second
        printf("%s\n", str);
    }
    return 0;
}
```

Simple Example(`cpu.c`): Code That Loops and Prints

Virtualizing the CPU (Cont.)

- Execution result 1.

```
prompt> gcc -o cpu cpu.c -Wall
prompt> ./cpu "A"
A
A
A
ˆC
prompt>
```

Run forever; Only by pressing "Control-c" can we halt the program

Virtualizing the CPU (Cont.)

- Execution result 2.

```
prompt> ./cpu A & ; ./cpu B & ; ./cpu C & ; ./cpu D &
[1] 7353
[2] 7354
[3] 7355
[4] 7356
A
B
D
C
A
B
D
C
A
C
B
D
...
```

Virtualizing the CPU (Cont.)

- Execution result 2.

```
prompt> ./cpu A & ; ./cpu B & ; ./cpu C & ; ./cpu D &
```

Even though we have only one processor, all four of programs seem to be running at the same time!

Virtualizing Memory

- The physical memory is *<u>an array of bytes</u>*.
- A program keeps all its data structures in memory.
  - **Read memory** (load):
    - Specify an address to be able to access the data
  - **Write memory** (store):
    - Specify the data to be written to the given address

Virtualizing Memory (Cont.)

- A program that Accesses Memory (`mem.c`)

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int
main(int argc, char *argv[])
{
    int *p = malloc(sizeof(int));  // a1: allocate some
                memory
    assert(p != NULL);
    printf("(%d) address of p: %08x\n",
        getpid(), (unsigned) p);  // a2: print out the
            address of the memmory
    *p = 0;  // a3: put zero into the first slot of the memory
    while (1) {
        Spin(1);
        *p = *p + 1;
        printf("(%d) p: %d\n", getpid(), *p); // a4
    }
    return 0;
}
```

Virtualizing Memory (Cont.)

- The output of the program `mem.c`

```
prompt> ./mem
(2134) memory address of p: 00200000
(2134) p: 1
(2134) p: 2
(2134) p: 3
(2134) p: 4
(2134) p: 5
^C
```

- The newly allocated memory is at address `00200000`.
- It updates the value and prints out the result.

Virtualizing Memory (Cont.)

- Running `mem.c` multiple times

```
prompt> ./mem &; ./mem &
[1] 24113
[2] 24114
(24113) memory address of p: 00200000
(24114) memory address of p: 00200000
(24113) p: 1
(24114) p: 1
(24114) p: 2
(24113) p: 2
(24113) p: 3
(24114) p: 3
...
```

- It is as if each running program has its **own private memory**.
  - Each running program has allocated memory at the same address.
  - Each seems to be updating the value at `00200000` independently.

Virtualizing Memory (Cont.)

- Each process accesses its own private **virtual address space**.
  - The OS maps address space onto the physical memory.
  - A memory reference within one running program does not affect the address space of other processes.
  - Physical memory is a shared resource, managed by the OS.

The problem of Concurrency

- The OS is juggling many things at once, first running one process, then another, and so forth.
- Modern multi-threaded programs also exhibit the concurrency problem.

Concurrency Example

- A Multi-threaded Program (`thread.c`)

```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        counter++;
    }
    return NULL;
}

int
main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: threads <value>\n");
        exit(1);
    }
```

Concurrency Example

- A Multi-threaded Program (`thread.c`)

```
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        counter++;
    }
    return NULL;
}
...
```

Concurrency Example (Cont.)

- The main program creates **two threads**.
  - Thread: a function running within the same memory space. Each thread start running in a routine called `worker()`.
  - `worker()`: increments a counter

```c
int
main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: threads <value>\n");
        exit(1);
    }
    loops = atoi(argv[1]);
    pthread_t p1, p2;
    printf("Initial value : %d\n", counter);

    Pthread_create(&p1, NULL, worker, NULL);
    Pthread_create(&p2, NULL, worker, NULL);
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("Final value : %d\n", counter);
    return 0;
}
```

Concurrency Example (Cont.)

- `loops` determines how many times each of the two workers will **increment the shared counter** in a loop.
  - `loops`: 1000.

```
prompt> gcc -o thread thread.c -Wall -pthread
prompt> ./thread 1000
Initial value : 0
Final value : 2000
```

  - `loops`: 100000.

```
prompt> ./thread 100000
Initial value : 0
Final value : 143012 // huh??
prompt> ./thread 100000
Initial value : 0
Final value : 137298 // what the??
```

Why is this happening?

- Increment a shared counter → takes three instructions.
    1. Load the value of the counter from memory into register.
    2. Increment it
    3. Store it back into memory

- These three instructions do not execute atomically. → Problem of **concurrency** happens.

Persistence

- Devices such as DRAM store values in a volatile place.
- *Hardware* and *software* are needed to store data persistently.
  - **Hardware**: I/O device such as a hard drive, solid-state drives(SSDs), etc.
  - **Software**:
    - File system manages the disk.
    - File system is responsible for storing any files the user creates.

Persistence (Cont.)

- Create a file `(/tmp/file)` that contains the string “hello world”

```c
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <fcntl.h>
#include <sys/types.h>

int
main(int argc, char *argv[])
{
    int fd = open("/tmp/file", O_WRONLY | O_CREAT
                 | O_TRUNC, S_IRWXU);
    assert(fd > -1);
    int rc = write(fd, "hello world\n", 13);
    assert(rc == 13);
    close(fd);
    return 0;
}
```

`open()`, `write()`, and `close()` system calls are routed to the part of OS called the file system, which handles the requests

Persistence (Cont.)

- What OS does in order to write to disk?
  - Figure out **where** on disk this new data will reside
  - **Issue I/O** requests to the underlying storage device

- File system handles system crashes during write.
  - **Journaling** or **copy-on-write**
  - Carefully <u>ordering</u> writes to disk

Design Goals

- Build up **abstraction**
  - Make the system convenient and easy to use.

- Provide high **performance**
  - Minimize the overhead of the OS.
  - OS must strive to provide virtualization <u>without excessive overhead</u>.

- **Protection** between applications
  - <u>Isolation</u>: Bad behavior of one does not harm others and the OS itself.

Design Goals (Cont.)

- High degree of **reliability**
  - The OS must also run non-stop.

- Other issues
  - Energy-efficiency
  - Security
  - Mobility

Break

What’s Next

- Take a look at course Canvas pages

- Programming Assignment 1
  - Due by Sunday, February 1 by 11:59pm ET
  - Write a UNIX Shell

- Module Software Preparation (HW0)
  - Due by Sunday, January 25 by 11:59pm ET
  - Install Virtual Machine (VirtualBox or UTM or something else)
  - Install GitHub repository
  - Verify C Compiler works
  - Push file into GitHub private repo

- Read UNIX History paper from Canvas (July 1974)
  - Anytime this week – Thompson and Ritchie