CS370: Operating Systems [Spring 2016]  
Dept. Of Computer Science, Colorado State University

### Topics covered in this lecture
- Course Overview
- Expectations
- Introduction

### Course webpage
- All course materials will be on the course webpage  
  http://www.cs.colostate.edu/~cs370
- Lectures
- Assignments
- Announcements
- Grades will be posted on Canvas
- There is also a wiki on the course webpage  
  FAQs and discussions for assignments
- The course website, Canvas, wiki, and Checkin are all live now

### Office Hours
- Instructor  
  Shrideep Pallickara  
  Computer Science (CSB 364)  
  Office Hours: 4:00-5:00 pm on Tuesday  
  9:00-10:00 am Friday
- GTAs  
  Swapnil Ashtekar  
  Meghana Santhapur  
  Office Hours in CSB 120: TBA
- All e-mail should be sent to cs370@cs.colostate.edu  
  The subject should start with CS370:

### Topics we will cover in CS 370
- Processes and Threads
- CPU Scheduling
- Process Synchronization
- Deadlocks
- Unix I/O
- Memory Management
- File System interface and management. Unix file system.
- Storage Management including SSDs and Flash Memory
- Virtualization

### Course Textbook
  Avi Silberschatz, Peter Galvin, and Greg Gagne  
  Publisher - John Wiley & Sons, Inc.  
When I make slides, I usually refer to several texts. These include …

- I always list my references at the end of every slide set

On the schedule page

- You will see the topics that will be covered and the order in which I will cover them
- The readings section describe the chapters that I will cover
- You will also see chapters from other books besides the textbook
- You will also see the schedule for when the assignments will be posted and when they are due

Grading breakdown

- Assignments: 45%
- Quizzes: 10%
- Mid Term: 20%
- Comprehensive Final Exam: 25%

Grading Policy I

- Letter grades will be based on the following standard breakpoints:
  - >= 90 is an A, >= 88 is an A-
  - >=86 is a B+, >=80 is a B
  - >=76 is a C+, >=70 is a C
  - >=60 is a D, and <60 is an F.
- I will not cut higher than this, but I may cut lower.
- There is no extra credit
- Any credit you earn, you must do so on a level-playing field with your peers
- There will be no make-up exams

Grading Policy II

- Every assignment (except HW0) will be posted at least 2 weeks before the due date.
- Every assignment will include information about how much it will count towards the course grade, and how it will be graded.
- Late submission penalty: 10% per-day for the first 2 days and a ZERO thereafter.
- Detailed submission instructions posted on course website.
- Assignments will be graded within 2 weeks of submission
If you are interested in taking this course with the honors option

- Honors courses are expected to be **tougher** courses
- You will be given 1 extra assignment
  - The best you can do on this assignment is get a 0
- You might have gotten an A in the regular course
  - But deductions in the extra assignment may result in you getting a lower grade

For the Quizzes and Tests

- I will only ask questions about what I teach
  - If I didn’t teach it, I won’t ask from that portion
- If the concepts were covered in my slides
  - You should be able to answer the questions
- I won’t ask questions about arcane aspects of some esoteric device controller

Exams

- There will be one mid-term (20%)
- The final exam is comprehensive (25%)
- There will be 12 quizzes
  - 2 quizzes where you had your lowest scores will be dropped
    - We will compute the average of your 10 highest scores
    - 10% of your course grade
  - If you walk into class more than 30 minutes late, there is an automatic 75% deduction on the quiz score.
    - There will be no negative scores.

Term project and poster session

- Raspberry Pi
  - 5 different deliverables
- There will also be a poster session where you will describe your work

Use of laptops, cell phones, tablets, and other electronic devices

- **Is not allowed**
- If you must use a laptop/tablet you will have to
  - Turn off wireless
  - And use it only for taking notes
- Authorized laptop/tablet users
  - Will sit in the back row starting at the corners

C Help Sessions

- Help session will be scheduled
  - Refresh C concepts
  - Voluntary participation: Primarily for students who have not programmed in C before.
- Computer Science Building: Room 130
  - Based on room availability this week
I do research in the area of distributed systems; these include

- Cloud computing
- Content dissemination systems
- Object Request Brokers
- Grid computing
- Peer-to-peer systems
- Collaborative systems

My research has been deployed in

- Commercial internet conferencing systems
- Defense applications
- Earthquake sciences
- Epidemic modeling
- Healthcare
- Bioinformatics
- Brain Computer Interfaces
- High energy physics
- Visualizations

Expectations

If you don’t have the discipline to focus, you surely won’t have the discipline to catch-up

- You will try to attend all classes
- You will focus on the discussions, and not on …
  - Other assignments
  - Social networking updates
- Assignments have to be done individually

What it takes to succeed

- You are required to work at least 6-8 hours per-week outside of class
  - Coding and reviewing material from class
- If you miss a lecture?
  - Add about 3 hours per missed lecture
How to fail this course?

- Believing that you can learn via osmosis
- **Missing lectures**
  - If you don’t have the discipline to show up, you will most likely not have the discipline to catch up
- **Procrastinating**
  - Get started on the assignments early

Please be kind to your peers

- No chatting in class please
- No eating in class please
- No cellphone use in class (even for texting)
- If you have signed the pledge and are using your laptop/tablet
  - Please sit in the last row of the classroom starting at the corners

Help me help you

- We will have **surveys** at the end of every class
- You will provide a list of
  - 3 concepts you followed clearly
  - 3 concepts you had problems keeping up with
- Problem areas for the majority of the class will be addressed in the next class.

Interactions

- You can have discussions with me, the GTAs, and your peers
- There are **two constraints** to these discussions
  - No code can be exchanged under any circumstances
  - No one takes over someone else’s keyboard
- Bumps are to be expected along the way
  - But you should get over this yourself
  - It will help you with the next problem you encounter

A modern computer is a complex system

- Multiple processors
- Main memory and Disks
- Keyboard, Mouse and Displays
- Network interfaces
- I/O devices
Why do we need Operating Systems?

- If every programmer had to understand how all these components work?
  - Software development would be arduous
- Managing all components and using them optimally is a challenge

Computers are equipped with a layer of software

- Called the **Operating System**
- Functionality:
  - Provide user programs with a better, simpler, cleaner model of the computer
  - Manage resources efficiently

A common misconception about the OS

- Is it the program that users interact with?
  - Text based: Shell
  - Graphical User Interfaces (GUI) that have icons etc.
    - The look-and-feel if you will
- This is **not actually part** of the OS
  - But it does use the OS to get its work done

Where the operating system fits in

- The OS runs on bare hardware in **kernel mode**
  - Complete access to all hardware
  - Can execute any instruction that the machine is capable of executing
- Provides the base for all software
  - Rest of the software runs in **user-mode**
    - Only a subset of machine instructions is available

The OS controls hardware and coordinates its use among various programs
Kernel and user modes

- Everything running in kernel mode is part of the OS
- But some programs running outside it are part of it or at least closely associated with it

Operating systems tend to be huge, complex and long-lived

- Source code of an OS like Linux or Windows?
  - Order of 5 million lines of code (for kernel)
    - 50 lines/page, 1000 pages/volume = 100 volumes
- Application programs such as GUI, libraries and application software?
  - 10-20 times that

Why do operating systems live for a long time?

- Hard to write and folks are loath to throw it out
- Typically evolve over long periods of time
  - Windows 95/98/Me is one OS
  - Windows NT/2000/XP/Vista/7/8/10 is another
  - System V, Solaris, BSD derived from original UNIX
  - Linux is a fresh code base
    - Closely modeled on UNIX and highly compatible with it
  - Apple OS X based on XNU (X is not Unix) which is based on the Mach microkernel and BSD's POSIX API

An operating system performs two unrelated functions

- Providing application programmers a clean abstract set of resources
  - Instead of messy hardware ones
- Managing hardware resources

The OS as an extended machine

- The architecture of a computer includes
  - Instruction set, memory organization, I/O, and bus structure
- The architecture of most computers at the machine language level
  - Primitive and awkward to program especially for I/O

Lets look at an example of floppy disk I/O done using NEC PD765

- The PD765 has 16 commands
  - For reading and write data, moving the disk arm, formatting tracks, etc.
  - Specified by loading 1-9 bytes into the device register
- Most basic commands are for read and write
  - 13 parameters packed into 9 bytes
    - Address of disk block, number of sectors/track, inter-sector gap spacing etc.
But that’s not the end of it …

- When the operation is completed
  - Controller returns 23 status and error fields packed into 7 bytes
- You must also check the status of the motor
  - If it is off? Turn it on before reading or writing
  - Don’t leave the motor on for too long
  - Floppy disk will wear out
  - TRADEOFF: Long start-up delay versus wearing out disk

Of course the average programmer does not want to have any of this

- What they would like is a simple, high-level abstraction to deal with
- For a disk this would mean a collection of named files
  - Operations include open, read, write, close, etc.
  - BUT NOT
    - Whether the recording should use frequency modulation
    - The state of the motor

Why do processors, disks, etc. present difficult, awkward, idiosyncratic interfaces?

- Backward compatibility with older hardware
- Desire to save money
- Sometimes hardware designers don’t realize (or care) how much trouble they cause!

Why abstractions are important

- Abstraction is the key to managing complexity
- Good abstractions turn a nearly impossible task into two manageable ones
  1. Defining and implementing abstractions
  2. Using abstractions to solve problem
- Example
  - File

Operating systems turn ugly hardware into beautiful interfaces

Two views of the operating system

- Top-down view
  - Providing abstractions to the application programs
- Bottom-up view
  - Manage all pieces of a complex system
The operating system as a resource manager

- Provide **orderly** and **controlled** allocation of resources to programs competing for them
  - Processors, memories, and I/O devices

The System view of the OS is that of a Resource Allocator

- An OS may receive numerous & conflicting requests for resources
  - Prevent errors and improper use
- Resources are scarce and expensive
- The OS allocates resources to specific programs and users
  - The allocation must be **efficient and fair**
  - Must increase overall system throughput

Defining Operating Systems

- Solves the problem of creating a usable computing system
  - Makes solving problems easier
- Control, allocate and mediate access to resources
- It is the one program that is running all the time: **kernel**

A (very) Brief History of Operating Systems

- Charles Babbage (1792-1871)
  - Spent most of his life and fortune trying to build the analytical engine
  - Never got it working properly
    - Purely mechanical
    - Technology of the day could not produce wheels, cogs, gears to the required precision
  - Did not have an operating system
Babbage realized he would need software for his analytical engine

- Hired Ada Lovelace as the world's first programmer
- Daughter of British poet Lord Byron
- The programming language Ada® is named after her

The First Generation (1945-55)
Vacuum Tubes

- First fully functioning digital computer built at Iowa State University
  - Prof. John Atanasoff and grad student Clifford Berry
- All programming in absolute machine language
  - Also by wiring up electrical circuits
  - Connect 1000s of cables to plugboards to control machine's basic functions
- Operating Systems were unheard of
- Straightforward numerical calculations
  - Produce tables of sines, cosines, logarithms

The Second Generation (1955-1965)
Transistors and Batch Systems

- Separation between designers, builders, operators, programmers, and maintenance
- Machines were called mainframes
- Write a program on paper, then punch it on cards
  - Give card deck to operator and go drink coffee
  - Operator gives output to programmer

The Third Generation (1965-1980)
ICs and Multiprogramming

- Managing different product lines was expensive for manufacturers
  - Customers would start with a small machine, and then outgrow it
- IBM introduced the Systems/360
  - Series of software-compatible machines
  - All machines had the same instruction set
  - Programs written for one machine could run on all machines

The Fourth Generation (1980-Present)
Personal Computers

- Large Scale Integration circuits (LSI)
  - Thousands of transistors on a square centimeter of silicon
- 1974: Intel came out with the 8080
  - General purpose 8-bit CPU
- Early 1980s IBM designed the IBM PC
  - Looked for an OS to run on the PC
  - Microsoft purchased Disk Operating System and went back to IBM with MS-DOS

COMPONENTS OF A COMPUTER
Components of a simple personal computer

- CPU
- Disk Controller
- USB Controller
- Graphics Adapter
- Memory
- Bus

Processors

- **Brain** of the computer
- Each CPU has a specific set of instructions that it can execute
  - Pentium cannot execute SPARC and vice versa

Rationale for registers inside the CPU

- Accessing memory to get instruction or data
  - Much longer than executing the instruction
- Registers hold:
  - Key variables
  - Temporary results

What the instruction set looks like

- Load a word from memory into register
  - And, from register into memory
- Combine two operands from register, memory, or both into a result
  - E.g. add two words and store result in a register or in memory

Besides the registers to hold variable and temporary results there are special registers

- **Program Counter**
  - Contains the memory address of the next instruction to be fetched
- **Stack pointer**
  - Points to the top of the current stack in memory
- **Program Status Word**
  - Stores condition code bits and other control code bits
  - Plays an important role in system calls and I/O
Memory

- Ideally the memory should be
  - Extremely fast: Faster than executing an instruction
    - CPU should not be held up by the memory
  - Abundantly large
  - Dirt cheap
- No current technology satisfies all these goals

Storage system hierarchy based on speed, cost, size and volatility

- Volatile: Registers, Main Memory, Electronic Disk
- Non-volatile: Cache, Optical Disk, Magnetic Disk, Magnetic Tapes

Memory Hierarchy: Registers internal to the CPU

- Made of same material as the CPU
  - Just as fast as the CPU
- Storage capacity is typically 32 x 32 bits on a 32-bit CPU
  - 64 x 64 bits on a 64-bit CPU
- Programs must manage registers in software

Memory hierarchy: Cache memory

- Mostly controlled by hardware
- Main memory divvied up into cache lines
  - Usually 64 bytes
  - Addresses 0-63 in cache line 1, 64-127 in cache line 2, and so on
- Most heavily used cache lines stored in high-speed cache close to the CPU

When a program needs to read a memory word

- Cache hardware checks if the needed line is in the cache
  - If it is, that's a cache hit
    - Request satisfied from cache in about 2 clock cycles
    - No memory access needed
  - If needed line is not present in cache
    - Cache miss, and must access memory
    - Substantial time penalty

Caching is a powerful concept used elsewhere too. Let’s see when ...

1. Large resource can be divided into pieces
2. Some pieces used more heavily than others
- OS caching examples:
  - Pieces of heavily used files in main memory
    - Reduce disk accesses
    - Conversion of file names to disk addresses
    - Addresses of Web pages (URLs) as hosts
CPUs usually have a couple of caches

- **L1 cache** is inside the CPU
  - Typically in the order of 16 KB
  - No access delay
- **L2 cache** holds several MB of data
  - Access delay of 1-2 clock cycles

Main Memory

- Usually called **RAM** (Random Access Memory)
- Cache misses go to the main memory
- **Volatile**
  - Contents lost when power is turned off
- Memory size is of the order of several GB in most modern desktops

Loading and storing of memory addresses is the precursor to processing

- `load()` moves word from main memory to an internal register
- `store()` moves content from register to main memory
- CPU automatically loads instructions from main memory

The instruction execution cycle

- Instruction **fetched** from memory and stored in instruction register
- Instruction is **decoded**, and operands fetched from memory and stored in some register
- Instruction on operands is **executed** next
- Result **stored** back in memory

Computers run most of their programs from (rewriteable) main memory

- Typically implemented in a technology called DRAM (dynamic random access memory)
- Ideal Scenario: Programs and data reside permanently in main memory. BUT ...
  - Space is **limited**
  - Main memory is **volatile** storage

The contents of this slide-set are based on the following references