CSCI 4061
Introduction to Operating Systems

Instructor: Abhishek Chandra

Outline

- Course Overview
- Operating System Definition
- OS Evolution
- OS Structure

General Information

- Lectures: Tu/Th 4.00-5.15 pm
- Course Web page: http://www-users.cselabs.umn.edu/classes/Spring-2014/csci4061
  - Also a moodle page (for forum, HW submissions, etc.)
- Office Hours:
  - Location: KHKH 4-209, Time: Tu Th 3-4pm
- Contact: email preferred
- Help email: csci4061-help@cs.umn.edu
  - Goes to instructor and all TAs
  - Use this as far as possible – likely to get faster response

TA Information

- Abhijeet Gaikwad, Ragavendra Natarajan, Gabriel Oliveira, and Hao Wen
- Will conduct recitation sections and office hrs
  - Details on course website
  - You can go to any TA’s office hrs
Course Objectives

- Learn Operating System Concepts
- A “Programmer’s view” of Operating Systems
- Focus on external OS interfaces and services
- Strong emphasis on systems programming
- Concepts applicable to non-Unix OS’s
  - Processes, File Systems, Virtual Memory
  - Concurrency and Synchronization
  - Networking and Sockets

Course Objectives (contd.)

- At the end of the course, you should be able to:
  - Understand OS structure, interfaces, utilities
  - Write readable and efficient programs
  - Use several OS APIs, tools, and libraries
  - Understand performance tradeoffs in user programs and the OS
  - Build better programs and systems:
    - E.g.: Web server, browser

Course Non-Objectives

- What you won’t be taught in this course:
  - C programming: You’ll have to pick it up yourself, though you would get some help in the initial labs
  - OS internals: This would be covered in 5103
  - Unix Tools: Some tools would be covered, but this is not a Unix tutorial course.

Course Non-Objectives (contd.)

- This course is NOT:
  - Kernel hacking course
  - User’s guide to Unix (though you will get some exposure to Unix tools)
Pre-Requisites
- CSCI 2021 (Computer Architecture): Requires good understanding of computer organization and hardware concepts
- Familiarity with Unix environment
- Good programming skills: C would be used in the course, but familiarity with another language such as Java would be helpful too
- Good understanding of data structures and algorithm fundamentals

Course Work and Mechanics

Recitation Sections
- Must attend recitation in addition to lectures
- TAs would conduct the recitations
- Discussion of course material
- Hands-on exercises
- Clarifications on Assignments

Class Discussion Forum
- On Moodle class site
- You can post questions, clarifications, discuss ideas, course material
- Try responding to each other as far as possible
- Instructor, TAs will regularly monitor the forum
- However:
  - No irrelevant, abusive mails
  - No posts that break the rules/spirit of honesty
  - Don't ask for solutions or post parts of your solution
  - Don't be too trigger-happy!
Textbooks

- Required:
  - "Unix Systems Programming", 2\textsuperscript{nd} Ed. by Robbins & Robbins

- Optional:
  - "Operating System Concepts", 9\textsuperscript{th} Ed. by Silberschatz et al.

Readings and Lecture Notes

- Weekly readings from textbook/external sources on website
  - Must keep up to follow lectures/recitations
- Lecture Notes would be available online before lecture (most of the time)
  - Print your own copies if you need hard copies
- Additional reading material would be online: Links on the class web-page

Course Requirements

- Readings from the book and assigned lecture notes and additional material
- 5 Programming Assignments (50%)
  - To be done in teams of 2
  - Each assignment due in 2 weeks
- Exams (50%)
  - 2 Mid-term Exams: In-class
  - Final Exam: 2 hrs

Programming Assignments

- Programs must be written in C
- The programs should be well-documented
- Provide full code, header files, makefiles, test-files, README file
- Online submission by 11:59 pm on due date (via Moodle site)
  - One submission per team
- Late submission policy:
  - 10% penalty for <24 hrs late
  - No submission allowed beyond that
Programming Assignments (contd.)

- **The code must be original**
  - Not copied or derived from the Web, from past offerings, other students, programmer friend, ...
- **No sharing of code across teams**
  - Team members should work together
  - Discuss and ask questions on class forum, from TAs or instructor
- Grading: Points for
  - Functionality and correctness
  - Code readability and documentation
  - Read specifications very carefully!

Exams

- Mid-term exams would cover the material of previous 4-5 weeks
- Final exam would be comprehensive 2-hour exam
- Open notes/open book
- No electronic devices allowed
- No sharing of books or other material

Grading Policy

- [93-100] A
- [90-93) A-
- [87-90) B+
- [83-87) B
- [80-83) B-
- [75-80) C+
- [70-75) C
- [65-70) C-
- [60-65) D+
- [50-60) D
- [0-50) F

Expectations from you

- Attend lectures and recitations regularly
  - Very important for success in course
- Keep up with weekly readings, exercises
- Start on assignments **early!**
  - Not going to be trivial, one-day affairs
  - Will run into bugs, problems, questions
- Class etiquette
  - Be attentive, respectful to others
  - Be involved: ask and respond to questions, participate in discussions
Academic Dishonesty

- What does it include?
  - Copying assignments, cheating on exams, plagiarism
  - Programming assignments: Code should be original (not copied or derived from the web, other teams or external sources)
- Can result in serious consequences:
  - Can range from 0 on assignment to F in class or worse
  - U requires report to Office of Student Affairs
- Take this issue very seriously
  - All parties involved in cheating (helper and helpee) will be considered equally culpable
- If unsure, just ask!

Disability Statement

- If you have, or think you have, a disability, contact Disability Services
- Please get a letter from DS for any special accommodation request on course work
- I will try my best to make the required accommodations

What is an Operating System?
**Computer System**

- **Hardware**
  - CPU, Memory, Disks, Devices
- **Operating System**
  - Processes, File System, Virtual Memory, Threads, Sockets
- **Applications**
  - User Programs
  - Shells, Tools and Utilities

**What is an Operating System?**

- User Programs
  - Shells, Tools and Utilities
- Processes, File System, Virtual Memory, Threads, Sockets
- CPU, Memory, Disks, Devices

**User's View**

- Extended Machine
- Simple abstraction of hardware resources
  - CPU -> Processes, Threads
  - Memory -> Virtual Memory
  - Disks -> Files
  - Network interfaces -> Sockets
- **Goal:** Simple, easy to use
- Less important: System performance

**Programmer's View**

- Resource Manager
- Efficient division of resources among multiple users, programs
  - Multiple processes on same CPU
  - Multiple files on the same disk
  - Multiple connections on same network link
- Arbitrate conflicting demands
- **Goal:** Maximum system performance
System View

- Control Program
- Handle different events, user inputs, etc.
  - User typing commands on keyboard
  - Bytes being read from the disk
  - Packets arriving on the network interface
- Multiple concurrent and asynchronous events
- **Goal:** Ensure correctness and fairness

Operating System Evolution

- How did the OS evolve?
- Generation 1: Mono-programming
- Generation 2: Batch Systems
- Generation 3: Multi-programming
- Generation 4: Personal Computers
- Generation 5: Distributed Systems
- Generation 6: Mobile Computing

Gen 1: Mono-programming (1945-55)

- Large computers (building-size)
- Mechanical relays, vacuum tubes
- Direct programming of hardware
- Basically no OS!
- Similar to a programmer writing raw assembly code


- Mainframes
- Programmers submit multiple jobs
  - Stack of punch cards
- Programs were batched and fed into the system
- Results were output after a few hours
- OS did some elementary job scheduling
- Similar to today’s supercomputers
Gen 3: Multi-programming (1965-80)
- Multiple programs executing in parallel
- Allow more efficient use of CPU and I/O
- Time-sharing: Variant of multi-programming
  - Fast time-multiplexing between multiple jobs
  - Each user gets single-user view
- OS performs resource management and control
- MULTICS (MULTiplexed Information and Computing Service) and UNIX

Gen 4: Personal Computers (1980-)
- Started as single-user systems
- Evolved into multi-user, multi-programming systems
- Main stress: Ease of use
- Evolution of GUIs
- OS provides simple abstractions and resource management
- DOS, Windows, MacOS, Linux, FreeBSD

Gen 5: Distributed Systems (1985-)
- Multiple computers linked by a network/bus
- Various flavors:
  - Multiprocessor systems
  - Client-server systems
  - Peer-to-peer systems and Grids
  - Cloud computing
- Focus: Interaction of multiple independent entities

Gen 6: Mobile Computing (2000-)
- Handheld devices: smart phones, tablets
- Main constraints: power, size/weight
- Limited resources:
  - Slower CPU, less memory storage
- Many sensors:
  - Cameras, GPS, accelerometers, ...
- Focus: energy-efficiency, mobility, context
- Examples: Google Android, Apple iOS
Other Operating System Flavors

- Embedded OS
  - Limited CPU, memory, battery
  - E.g.: Home devices, environmental sensors
  - Special-purpose functionality
- Real-time OS
  - Time-based guarantees
  - E.g.: Space rockets, cars, production machinery
- Multimedia OS
  - Soft real-time requirements
  - E.g.: Audio/video servers