EECS 338: Introduction to Operating Systems and Concurrent Programming

Spring 2016, http://cwru-eecs338.github.io/

Instructor: Eamon Johnson

Office: TBA, e-mail: eamon@case.edu

Class Hours and Place: **Lectures:** M-W: 4:00pm-5:15pm, Place: Bingham 103
Recitation: Tue: 6pm-7pm, Place: Bingham 103

Office Hours: TBA

Prerequisites: EECS 233, or the consent of the instructor.

Course web page: http://cwru-eecs338.github.io/

Main course textbook: *Operating System Concepts*, 9th edition, Windows XP update, John Wiley, 2013, ISBN: 978-1-118-06333-0 (You can rent this book for $42.84 from amazon.com) Also: 8th edition is available online here.

Linux Textbook: Interprocess Communication in Linux, J. S. Gray, 3rd edn, Prentice-Hall, 2003. This book is available online here.

Coverage: Chapters 1-9, 11-13, 16, 17 and notes. Please see the web page for details.

Revised Course description in the University Bulletin (as of Spring 2016**):**

- **Intro to Operating Systems:** OS structures, processes, threads, CPU scheduling, deadlocks, memory management, file system implementations, virtual machines, cloud computing.
- **Intro to concurrent programming**: fork, join, concurrent statement, critical section problem, safety and liveness properties of concurrent programs, process synchronization algorithms, semaphores, monitors.
- **Intro to UNIX systems programming:** UNIX system calls, UNIX System V IPCs, threads, RPCs, shell programming.

TAs: Emilio Colindres exc231@case.edu
Yuanxu Li yuanxu.li@case.edu
Xiaojin Li xxl509@case.edu

Quiz and Midterm Exam Rules:

All quizzes and the midterm exam are in-class.

I must have a priori notice for any quiz or the exam missed. There will not be any make-up quizzes.

Quiz, Exam, Final Exam Dates:

All quiz coverage areas are tentative, and subject to change during the semester.

Midterm Exam:	Concurrent Algorithms
Quiz 1:	Chapters 1, 2, and 3 (Intro, OS Structures, Processes)
Quiz 2:	Chapters 4, 6 (Threads, CPU Scheduling)
Quiz 3:	Chapters 7, 8, 9 (Deadlocks, Memory Management, Virtual Memory)
Quiz 4:	Chapters 11, 12, 10, 16 (File systems, Disk Scheduling, Virtual Machines)

There will not be a final exam in this course.

Grading:

Assignments: 50%

All assignments require Linux and C use.

Your code must run on the department's virtual (Ubuntu) Linux machine for the grader to run and test your code.

All assignment docs are to be submitted as either pdf or zipped files: no txt, rtf, doc, etc, files.

Quizzes: 25% (Duration: 25 minutes each)

Midterm Exam: 25% (Duration: 1hr 15 minutes)

Late Assignment Policy: Late assignments will not be accepted. However, each student is allowed to have either one **extension** of at most 48 hours, or two extensions each of at most 24 hours. You are to enter your extension request into a google sheet (to be announced) before the deadline of the assignment.

Assignment re-grading policy: Once a grade is posted, you have one week to request a re-grading of an assignment by entering your name to a google sheet (to be announced).

DETAILED COURSE DESCRIPTION

EECS 338 helps students gain an understanding of three essential computer science areas:

a. Introduction to operating systems,
b. Introduction to concurrent programming,
c. Introduction to UNIX systems programming on items a and b above, that eventually ends at multi-threaded programming—an essential topic, given today's multi-core CPU computers.

Items a and b are covered solely in the lectures; item c is covered mostly in recitations, and also in lectures. Note that EECS 338 is a 4-credit hour course, and with a weekly recitation. Below is a more detailed description of what EECS 338 covers.

LECTURES:

1. ***Operating Systems:***
   - ***(i) Basics of operating systems (OS)***
     a. OS Structures
     b. Processes
     c. CPU scheduling
     d. Deadlocks
     e. Memory management and Virtual Memory Management Issues
     f. File System Interfaces and File System Implementation.
     Also, at the end of the course, if time permits, as a natural extension of item f, one or two of the contemporary "big data" file systems from among Hadoop, GFS (Google file system), and Google's Map-Reduce are briefly covered.
   - ***(ii) Introduction to more advanced OS topics.***
     g. Virtual Machines: This part uses knowledge gained in all the sub-topics covered in part 1(i).
     Virtualization is the starting point for Cloud Computing, an area still evolving, but one of utmost importance to computer science.
        i. Type 0-3 hypervisors and their design/implementation details

ii. Techniques that make virtual machines efficient and effective (trap-and-emulate; binary translation, para-virtualization, emulators, app containment, etc.)

h. Very basic intro to Cloud Computing--without any of the underlying "as-a-service" data management/virtualization technologies such as software-as-a-service, platform-as-a-service, DBMS-as-a-service, infrastructure-as-a-service, etc.
Computer science students need to know what Cloud Computing is, at a technical level, as this is now an important part of computing and computer science.

2. ***Concurrent Programming Basics:*** This part prepares students for multi-threaded programming—now a necessity in the world of multi-core machines. Parallel programming (needed for cloud computing) cannot be fully understood unless one knows concurrent programming basics.
    a. Models of computation for concurrent processes: shared memory versus message passing models.
    b. Concurrent programming basics: fork, join, the concurrent statement of Dijkstra; nondeterministic behavior or concurrent programs (order nondeterminism), determinate versus indeterminate programs,
    c. Critical section problem: mutual exclusion, bounded waiting, progress properties. Safety and liveness properties of concurrent programs (and a very brief mention of how they relate to Hoare Logic and Temporal Logic, and program verification, without the formalizations/theory).
    d. Modularization of concurrent programs,
    e. Brute-force synchronization algorithms without additional concurrent programming constructs: Peterson's 2-process algorithm, n-process synchronization bakery algorithm, etc.
    f. Hardware solutions to synchronization: test-and-set instruction, swap instruction,
    g. Synchronization via binary or nonbinary semaphores,
    h. synchronization via critical region statements—only to be mentioned,
    i. Synchronization via monitors,
    j. Threads
        i. Thread types, basics, thread-safety, etc.,
        ii. POSIX thread management calls (also covered in recitations),
        iii. Multi-threaded programming via POSIX threads (also covered in recitations),
    k. Many concurrent programming problems and solutions to some of them (e.g., producer/consumer problem with single/multiple producers/consumers), readers-writers problem (many variants), dining philosophers problem, and many other concurrent problems that are defined and solved by T. Ozsoyoglu, such as the checking account problem, H_2O generation problem, etc..

3. Introduction to UNIX/Linux systems programming
    a. Basic Unix system calls such as fork(), exec(), exit(), wait(), getpid(), etc..
    b. Unix System V shared memory and semaphore IPC calls (InterProcess Communication)
    c. Unix System V message passing IPC calls.
    d. POSIX multi-threaded programming.
    e. RPCs (Remote Procedure Calls).
    f. UNIX shell creation.

RECITATIONS AND ASSIGNMENTS:

Recitations and assignments work in tandem to prepare the class to use Linux virtual machines, and to learn the Linux systems programming environment so that the class completes concurrent/multi-threaded programming assignments.

Recitations: See EECS 338 recitation web page: http://cwru-eecs338.github.com

Assignments*:* There are both programming and algorithmic assignments. Please see the web page for earlier years' assignments and exams.

Programming assignments: There are 4-5 Linux programming assignments designed to teach students Linux "systems programming" as well as Linux concurrent programming, chosen from:

1. A basic Unix/Linux system calls assignment involving, say, fork(), exec(), exit(), wait(), getpid(). This assignment allows students to create multiple processes and crudely manage them.
2. Linux pipe programming assignment
3. linux-shell building assignment from scratch (a parser program for command-line input is provided to the class).
4. A concurrent programming assignment that uses Unix System V shared memory and semaphore-based concurrent programming.
5. A concurrent programming assignment that uses POSIX multi-threaded programming. There are two variants, namely, posix semaphore-based programming, or posix lock (mutex)-based programming.
6. Concurrent programming via Linux Remote Procedure Calls (RPCs).

Algorithmic assignments**:** Algorithmic assignments prepare students to think algorithmically about thread-safety issues, and multi-threaded programming. (Multi-threaded programming assignments are about converting these algorithms into, usually C, programs.) There are two paper-and-pencil concurrent programming algorithm design assignments. In the past, students were taught how to solve concurrent problems via either semaphores, (conditional) critical region statements (CCRs), or monitors. Starting Spring 2015, CCR coverage has been removed, and, the focus is only on semaphore- and monitor-based solutions.