EECE 7376 – Operating Systems: Design and Implementation – Spring 2017

Instructor	Prof. Rafael Ubal
Email	ubal@ece.neu.edu
Website	http://ece.neu.edu/~ubal
Office	411 Dana Research Center
Phone	617-373-3895
Course calendar	https://goo.gl/jBf5ku
Office hours	The schedule for office hours can be found in the course calendar. Additional office hours are available by appointment. When attending office hours, please show up at the beginning of the designated time slot, since several students may have the same questions.
Teaching assistants	Nicolas Agostini (bohmagostini.n@husky.neu.edu) Trinayan Baruah (baruah.t@husky.neu.edu)

Overview

This course covers the fundamentals of operating systems (OS) design, including theoretical, OS-generic design considerations, as well as practical, implementation-specific challenges in the development of a real OS.

The course is organized in two parts, each accompanied with a separate course project developed by students, individually or in groups. The first part deals with the system call interface between an application and the OS, the multi-process abstraction of a computing system, multi-threading libraries, and thread communication and synchronization techniques. During this part, students will get familiar with the Linux system call interface through the implementation of a command-line interpreter (Unix shell).

The second part of the course covers memory management, file systems, disk I/O, protection, and virtualization. Students will engage in active implementations and extensions of a miniature OS, developed and debugged on an x86 processor emulator, and compatible with a real x86 PC. After completing this course, students will have a first-hand experience in the development of low-level software managing modern commercial processors.

Course Objectives

- Understand the architecture of a multi-task, multi-user system, composed of the system hardware, the operating system, and the application software, together with the interfaces between them.
- Learn how multiple applications can share processing resources, and how scheduling policies affect global system performance.
- Understand how a system's physical memory is distributed across the user-level applications transparently to the programmer.
- Learn how user-level applications communicate with hardware resources through the operating system services.
- Understand how a disk device is organized to provide a secure, fault-tolerant, and intuitive view based on files and directories.
- Understand the implications in security of a multi-user system, and the motivation for protection mechanisms.
- Get familiar with the Linux system call interface through the implementation of a command-line interpreter that spawns child processes and executes built-in commands.
- Get familiar with the core features of an OS kernel through hands-on implementation of new features on a miniature OS compatible with a modern x86 PC.

Prerequisites

This course relies on a proficient knowledge of the C programming language, and the GNU tool set for C programming and debugging on Unix operating systems. Experience is recommended in the use of a Unix operating system (such as Linux or OS-X) at a user level, and the use of basic shell commands (`ls`, `cd`, `ssh`, `gcc`, `gdb`, ...). Although a lack of user-level experience in Unix can be easily overcome during the first weeks of the course, strong prior C programming skills are indispensable to complete every homework and project assignment. The following book is a recommended review material:

- B. Kernighan, D. M. Ritchie, "The C Programming Language", ISBN 007-6092003106

References

Most of the material presented in class is based on the references below. These references can be considered as recommended reading. All material strictly needed to complete the course will be available through class notes, or as additional electronic handouts on Blackboard.

- R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau , “Operating Systems: Three Easy Pieces ”, May 2014 (Version 0.8) , http://www.ostep.org
- A. Silberschatz, P. B. Galvin, G. Gagne, "Operating System Concepts", ISBN 978-0470128725
- T. Anderson, M. Dahlin, "Operating Systems: Principles and Practice", ISBN 978-0985673512
- R. Love, "Linux Kernel Development", ISBN 978-0672329463
- Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 2 - Instruction Set Reference

Grading

Weights

• Homework	20%
• Quizzes	20%
• Midterm exam	20%
• Final exam	20%
• Course project	20% (+5% extra credit)

Homework assignments

There will be a total of 10 weekly homework assignments. Assignments will be posted on Blackboard at least 7 days before their due date, and must be submitted on Blackboard as well. Each homework assignment will require you to upload a PDF file with your answers, which you can produce from most common word editors (Microsoft Word, LibreOffice, LaTeX, …).

Homework due dates are strict deadlines with no exceptions. The exact due dates are specified at the end of this document. Late homework will not be accepted under any circumstances. Please make sure that you submit your assignments in advance in order to avoid unexpected submission problems due to Internet connectivity issues, trouble with PDF document generation, problems with the submission link, etc.

To add some flexibility to this policy, the average grade for homework assignments will be calculated by discarding either that which received the lowest grade or which was not submitted on time at all. This exception is aimed at covering any inevitable situation that prevented you from submitting a homework assignment on time, while it also benefits those students with no missing assignment.

Midterm and final exams

A midterm exam will cover the first part of the course material. A comprehensive final exam will focus on the second part of the course, but will also include the material corresponding to the first part. The date of the midterm is available at the end of this document. The date of the final exam will be announced during the second half of the semester.

Quizzes

There will be a total of 4 quizzes during the semester, on the dates specified in the schedule at the end of this document. Quizzes will have an approximate duration of 20 minutes, and will start in the beginning of the lecture time.

Course projects

Students will work on two projects throughout the course, either individually or in groups of at most two people:

- The first project consists in the implementation of a Unix shell, that is, a command-line interpreter for the user to interact with the OS services. The shell will support the execution of external commands by spawning child processes, as well as the execution of internal, built-in commands (`cd`, `echo`, `exit`, ...). This assignment will be due by the date of the midterm exam, announced in the beginning of the course.
- The second project consists in extending an open-source miniature operating system (*xv6*) with new functionality. The new features should be fully functional and tested on the QEMU full-system simulator. Even though you will be using the emulator for convenience, your OS extensions would also be able to run on a real machine. This assignment will be due by the date of the final exam, which will be announced during the second half of the semester.

Homework assignments will include the development of some modules that you will be able to reuse in the course projects. You can earn up to 5% extra credit with outstanding course projects. Additional features can be either inspired in the suggestions from open-ended homework assignments, or based on ideas of your own. Extra features in your project will be graded based on creativity, novelty, usability, and coding style.

You can also choose to give a 10-minute presentation on your project during the last lecture. Due to time limitations, a selection of volunteering presenters might be needed, which will be based on the quality of the projects.

Grade conversion

Your final grade is calculated as a numeric grade between 0 and 100 based on the percentages shown above, and then converted into a letter grade using the following scale:

High	Low	Grade	High	Low	Grade	High	Low	Grade
100	95	A	84.99	80	B	69.99	65	C
94.99	90	A-	79.99	75	B-	64.99	60	C-
89.99	85	B+	74.99	70	C+	59.99	0	F

Attendance and Punctuality

While attendance to the lectures is highly recommended, punctuality in class is indispensable, and constitutes a basic rule of respect toward your class mates and myself. If any particular reason forces you to come in late to class, please let me know in advance by email or in person.

If you come to office hours, please show up at the beginning of the corresponding time slot. It is expected that multiple students have similar questions and benefit from the same answers. This will allow us to use everyone's time more efficiently.

Teaching Assistant

Responsibilities

The Teaching Assistant has the following responsibilities:

- Grading the 10 homework assignments
- Grading the 2 class projects
- Grading the 4 quizzes
- Holding office hours by appointment

Office hours

The Teaching Assistant will be available for office hours by appointment, as requested by students through an email. The Teaching Assistant's availability will not include University holidays, such as Spring break, or Thanksgiving break. Students are encouraged to use the Teaching Assistant's office hours to discuss both content and grading of assignments, as well as to review any part of the class material.

Grading report

For each grading assignment, the Teaching Assistant will submit a grading report to the instructor no later than one week after the assignment submission deadline for the students. For example, assuming that an assignment is due on Feb 2 at 11:59pm, the grading report must be submitted by the Teaching Assistant no later than Feb 9 at 11:59pm.

A grading report should be sent to the course instructor in plain text in the body of an email. Each grading report should be sent in a separate email. The subject of a grading report should be formed of the course code in square brackets, the literal text "*Grading report -* ", followed by the assignment name. For example:

```
[EECE7376] Grading report - Homework 1

[EECE7376] Grading report - Quiz 2
```

The body of the email containing a grading report should have the following sections—notice that this is confidential information between the Teaching Assistant and the course instructor:

1. Best grade, worst grade, and average grade.
2. Common mistakes from the students, including suggestions of material that should be reviewed or emphasized in class by the instructor.
3. Suggestions on how to modify, extend, or rephrase the assignment to improve clarity and fairness. These suggestions will be considered by the instructor in the design of future assignments.
4. Instances of possible plagiarism. As soon as the Teaching Assistant senses a slight hint of plagiarism, the corresponding assignment should be cited here. The instructor will later review each case individually.

Grading format

Use the following guidelines to grade paper-based assignments:

- Always use permanent <u>red ink</u> for your comments and grades.
- Whenever points are deducted in an answer, it should be made clear why those points where deducted, as well as how many points were deducted for each mistake. Write the number of deducted points in parenthesis followed by a clear explanation of the mistake.

Each question will show how many points it is worth. When you finish grading it, write the number of points earned by the students on the left of the question title, and underline this information. Here is an example:

<u>3</u> **Question 1 (5 pt.)**
Write a Verilog module that describes the behavior of a circuit adding two 8-bit numbers. Use a dataflow model.

```
module Adder(input a, input b, output reg c);
      assign c = a + b;
endmodule
```

(−2)
Inputs and outputs should be 8 bits

- When you finish grading all questions in an assignment, calculate the total number of earned points, and circle the final grade on the top left corner of the first page, by the title of the assignment:

8

Name: John Smith

Quiz 1

<u>3</u> **Question 1 (5 pt.)**
Write a Verilog module that describes the behavior of a circuit adding two 8-bit numbers. Use a dataflow model.

Assignment pick-up and drop-off

Assignments involving physical hand-outs on paper, such as quizzes, will be picked up by the Teaching Assistant at the end of the class when the assignment was given, and at the same location where the lecture is held. The Teaching Assistant should be present at least 5 minutes before the lecture ends by the classroom's door, ready to pick up the material as soon as class ends.

Similarly, the Teaching Assistant is responsible for dropping off the graded material one week after the assignment was picked up, at the latest. The drop-off process is identical: the Teaching Assistant should be ready at least 5 minutes before class ends, and should come in as soon as the instructor finishes class. The instructor will give the assignments back to the students before they leave the classroom. The Teaching Assistant should feel free to drop off graded assignments at the end of a previous lecture, in those cases where the assignments were graded in less that one week.

Course Topics

C Language Review

- Building a program with *gcc*
- Basic types in C
- Strings
- Pointers
- Dynamic memory
- Data structures

Unit 1 – Introduction

- Organization of a computing system
- Properties of an OS: CPU virtualization, memory virtualization, concurrency, persistence

Unit 2 – CPU Virtualization

- Mechanism (time sharing) and policy (scheduling)
- The process
- The `man` pages
- The process image
- Process states: ready, blocked, running, zombie
- The `fork` system call
- The `wait` system call
- The `exec` system call
- The file table
- The `open` system call
- Input and output
- Input/output redirection: operators `<` and `>`

- Pipes
- The `pipe` system call
- The `dup` system call
- UNIX signals
- The `signal` system call
- The `kill` system call
- The CPU virtualization mechanism
- Limited direct execution

Unit 3 – Scheduling

- Scheduling policies
- FIFO (First In First Out)
- SJF (Shortest Job First)
- STCF (Shortest Time-to-Completion First)
- RR (Round Robin)
- Incorporating I/O
- The Multi-Level Feedback Queue (MLFQ)

Unit 4 – Memory Virtualization

- Goals: transparency, efficiency, isolation, protection
- Interface
- Mechanism: hardware-based translation, software-based configuration
- Segmentation: allocation, translation, segfault
- Memory waste: internal and external fragmentation
- Free space management: OS and LibC-level, linked list based implementation
- Free space management policies: best fit, worst fit, first fit, next fit
- Paging, page table storage, translation, comparison with segmentation
- Multi-level page tables: translation
- Translation-lookaside buffers (TLBs)
- Swapping

- The page fault

Unit 5 – Concurrency

- Threads
- Data sharing: race condition, critical section, mutual exclusion, atomic execution
- Locks
- Spinlocks
- The test-and-set instruction
- The compare-and-swap instruction
- OS support for locks: yielding the CPU
- Concurrent data structures: concurrent counter
- Calculation of *pi* using the Monte Carlo method
- Condition variables
- The producer/consumer problem
- Semaphores
- The dining philosophers
- Deadlock, starvation, livelock

Unit 6 – Persistence

- UNIX users and groups
- The UNIX file system
- Structure of a file system
- Inodes
- Directories
- Accessing a file
- Hard links
- Symbolic links
- Accessing file systems: device names in `/dev`, partitions, making/mounting file systems
- System calls to manage files: `open`, `close`, `read`, `write`, `lseek`, `stat`, `link`, `symlink`, `unlink`

- System calls to manage directories: `mkdir`, `rmdir`, `opendir`, `closedir`, `readdir`
- Crash consistency in file systems
- Recovery solution #1: the file system checker
- Recovery solution #2: journaling

Unit 7 – The xv6 Operating System

- Virtual machine monitors (VMMs)
- Introduction to x86 assembly: register, *mov/push/pop/call/ret* instructions, memory accesses, arithmetic, flags, branches
- Paging support in x86
- The boot loader: `bootasm.S`, `bootmain.c`
- Entry to xv6: `entry.S`
- Initial page table
- Processes: `struct proc`, `enum procstate`, `ptable`, `struct cpu`
- The kernel stack
- The first process: `proc.c`, `main()`, `userinit()`, `allocproc()`
- The context switch: `swtch()`
- The `init` program: `initcode.S`, `init.c`
- The memory manager, the 3 kernel page tables (`entrypgdir,` `kpgdir,` `p->pgdir`), physical memory allocation (`struct run`, `kmem`)
- New memory spaces: `setupkvm()`, `mappages()`, `walkpgdir()`, `switchkvm()`
- The user memory space: `allocuvm()`, `deallocuvm()`
- The `exec()` system call: `exec.c`
- The trap table: `tvinit()`, `idtinit()`, `vectors`, `trap()`
- Hardware interrupts, the timer interrupt: `lapicinit()`, `trap()`, `yield()`
- Device drivers, the disk driver: `struct buf`, `ideinit()`, `idewait()`, `iderw()`, `idestart()`, `ideintr()`

Access to COE Linux Computers

The College of Engineering provides a set of Linux machines available for students in the Computer Center at 271SN (Snell Engineering). These machines can be accessed remotely through an SSH client, or physically in the lab. No matter which machine you choose, you will see the same files in your home folder. Information about the computer lab is available at

http://www.coe.neu.edu/computer/

Notice that the COE account information is not the same as your MyNEU account. If you do not have a COE account yet, you need to request it by following the instructions on the link given above.

You can access a COE Linux machine by connecting to `gateway.coe.neu.edu` with an SSH client installed on your machine. On Windows, you can download *Putty*, a simple and lightweight interactive SSH client. On Linux/Mac, open a terminal and use command `ssh`.

In order to transfer files between your local machine and `gateway.coe.neu.edu`, you need to use an SFTP client on your local machine. On Windows, you can download tool *WinSCP*. On Linux/Mac, open a terminal and use command `scp`.