Course Syllabus

ENEE 447: Operating Systems — Spring 2021
Prof. Bruce Jacob

Basic Information

Time & Place

Lecture: TuTh 11:00am – 12:15pm, on-line — All Lab Recitations: on-line?

Section 0101 Lab: Mon 12:00pm – 1:50pm	*TA: Arpit Bansal, arpitbansal297@gmail.com*
Section 0102 Lab: Mon 2:00pm – 3:50pm	*TA: Taeyoung An, smilety@umd.edu*
Section 0103 Lab: Wed 12:00pm – 1:50pm	*TA: Arpit Bansal, arpitbansal297@gmail.com*
Section 0104 Lab: Wed 2:00pm – 3:50pm	*TA: Taeyoung An, smilety@umd.edu*

Professor

Bruce L. Jacob: AVW-1333, blj@umd.edu
Office hours: none until campus is opened up, mask-free

Class Home Page

http://classweb.ece.umd.edu/enee447

Class Piazza Link

piazza.com/umd/spring2021/enee447

Class Schedule

This is a weekly schedule of my hours, including class time and scheduled office hours, but also including other things that make me unavailable. It is subject to change.

	MON	TUE	WED	THU	FRI
9–9:30					
9:30–10					
10–10:30					
10:30–11					
11–11:30					
11:30–12		**ENEE 447** Lecture on-line		**ENEE 447** Lecture on-line	
12–12:30					Weekly meetings with collaborators and graduate students
12:30–1					
1–1:30					
1:30–2					
2–2:30					
2:30–3					
3–3:30					
3:30–4					
4–4:30					
4:30–5					

Course Overview

This course covers the design and development of operating systems and how they interact with the hardware on which they run. We will cover concepts such as multicore processors, interrupts and timers, coherence and multiprocessing, interprocess communication, multiple privilege levels and virtualization, processes and threads and context switching, virtual memory, permanent storage, flash memory systems, and more. The course is intended to give you a solid understanding of how operating systems are implemented today, but more importantly how they will be implemented tomorrow: it turns out that several advances at the hardware level (multicore and nonvolatile memories, in particular) now render moot a number of operating systems designs from the past.

You will learn the course concepts by not only reading about them but by building them; you will help design and then build a working operating system, from the ground up, on top of bare-metal hardware. The operating system will be written in assembly code and C (mostly C), and it will run on the popular *Raspberry Pi* platform. Because of the importance of multicore architectures on system-level programming, we will be focusing exclusively on the **Raspberry Pi 3B+**, because it has a quad-core, 64-bit ARM processor in it (Broadcom BCM2837, Cortex-A53). Programming for multiple cores is extremely challenging, and so it will be one of the main areas of focus in this course.

Building actual code on actual multicore hardware is interesting for several reasons. First, you must be *infinitely* more precise in your design than if you built your software for a simulator platform. Among many other differences, simulators only *emulate* simultaneity, they don't actually *do* it. Thus, they cannot generate real race conditions the way actual hardware does. This is good because programming for real multicore hardware will force you to understand all the finer points of your design and code implementation, as well as the ramifications of all your choices—if you are not thorough, it will not work. Second, real hardware implements numerous features that hardware simulators either can't do (such as simultaneity) or don't do, such as offer multiple protection levels. This is becoming extremely popular in datacenters, where it is simpler to run processes in virtual OS bottles than it is to run them directly on the main OS. These virtual OSes should have access to mechanisms unavailable in user mode, but they should not have free run of the machine. This is where multiple protection domains, such as hypervisor modes, come in. Knowing how these work and how to program for them is an advanced skill. Lastly, it's just kinda cool to have a working operating system that you built, running on a computer that you can hold in your hand.

Prerequisites

Students must have taken ENEE 350, or have equivalent knowledge of computer organization. You should understand what the program counter is and how it works, what the register file is and how it works, what a cache is and how it works, what memory is and how it works, etc. You must understand what assembly code is and how it works. Students should also have taken CMSC 330 and 351 and be adept in code development. You should understand and be extremely fluent in programming in C, e.g. using arrays, structures, functions, and pointers. The C language is particularly useful for our purposes in this class, as C was designed and written specifically to enable the development of an extremely powerful (and, now, popular) real-time operating system: Unix.

Course Materials

The course has the following required materials:

Operating Systems: Three Easy Pieces, by Arpaci-Dusseau & Arpaci-Dusseau
- textbook is available for free in PDF at http://pages.cs.wisc.edu/~remzi/OSTEP/
- hardcopy can be purchased at lulu.com

Raspberry Pi 3 Model B+, containing a 64-bit, 4-core Cortex-A53 ARM processor
- lots and lots and lots of info at https://www.raspberrypi.org
- board can be purchased for $35 at numerous sites
- you will also need a micro-SD card and SD adapter (look for RPi's NOOBS)
- lastly, you will need a USB-serial cable (also called a "console cable") (see https://learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable)

You must purchase your own RPi3 board, console cable, and Class-10 SD/microSD card (it is important that it be Class 10), so that you can do code development on your own, whenever and wherever you want. If you do not have an SD card reader built into your laptop, you will need to purchase your own USB-based card reader.

In addition, the following book is not required but is *highly* recommended, as it is brilliantly written (very dense, lots of information in a small space, very thin book) by the guys who invented the language:

The C Programming Language (2nd Ed.), by Kernighan & Ritchie

This is an invaluable book, and every serious programmer I know has a worn-out copy of it.

Another very useful book is the following in-depth treatment of the ARM architecture, written by designers at ARM:

ARM System Developer's Guide, by Sloss, Symes, and Wright

This will help you better understand the ARM platform and how to write low-level software for it.

Everything else will be handed out in class and posted on the course website.

Class Projects

A number of projects will be assigned during the term, each of which will require a *substantial* time commitment on your part. You will find the work load in this course to be extremely heavy. The projects will build upon each other, so you will need to keep up in order to finish a working OS by the end of the semester. Here is the tentative list of projects:

- Project 0: Purchase Raspberry Pi 3B+ board
  Get `aarch64-elf` cross-compilation environment up and running
  Design and build a timer facility
- Project 1: Purchase USB-serial "console cable" and install driver
  Get the echo server up and running
- Project 2: Design and build a timeout queue facility (i.e., the Unix *callout table*)
- Project 3: Implement interrupts & vectors in a single monolithic kernel
- Project 4: Design and build an inter-process communication facility
- Project 5: Implement context switch in isolation
  (using a rudimentary scheduler: a timed swapper)
- Project 6: Implement system-call facility for user-level applications
  Implement master-slave forced distributed context switch
  (core0 interrupts cores 1&2, moves running user app from core1 to core2)
- Project 7: Implement virtual memory in isolation
- Project 8: Implement multiple security levels (user-level, hypervisor, system)

- Project 9: Design and implement "full" suite of system calls
- Project 10: Access flash memory in isolation
- Project 11: Implement permanent objects, combined VMFS
- Project 12: Final operating system, including multiple security levels, multiple user-level applications, scheduling and context switching, real-time interaction, flash-based memory system with permanent objects [and hot restart?]

If it is not abundantly clear, this represents an *enormous* amount of work. The most common reason for not doing well on projects is not starting them early enough. You are given plenty of time to complete each project. However, if you wait to start until the week it's due, you **will not** be able to finish. Plan to do some work on a project every day. Also plan to have it finished several days ahead of the due date—many, many, many unexpected problems arise during debugging. You will find that this is ***not*** normal code development. **Plan** for this to happen. Your lack of starting early is **not** an excuse for turning in your project late, even if unfortunate situations arise such as lost SD cards, dead laptops, etc.

There are many sources of help on which you can draw. Simple questions can be submitted to the professor and fellow classmates via email (**use the email list given on page 1**). These will typically be answered within the day, often more quickly during working hours. Keep in mind, however, that many types of questions cannot be answered without seeing your project. If you have detailed questions, your best option is to speak to the TA or professor in person during office hours. Bring along a listing of your project and your SD card & RPi board. Students are also encouraged to help one another. *One of the best ways for you to make sure that you understand a concept is to explain it to someone else.* Keep in mind, however, that you should not expect anyone else to do any part of your project for you. The project that you turn in must be your own.

When Projects Are Due

Projects are assigned on Tuesdays and will be due during the recitation labs: they will be demonstrated by you to the TA during your lab/discussion section. At that time, you will explain and demonstrate your code to the TA. So that the later sections have no time advantage, *all projects will be due before **midnight** on Sunday, via the **submit** facility*. That means you must submit all of your work ***on or before 11:59 pm Sunday night***, prior to the Monday morning lab at which the first students will explain their projects to the TA. Please note: 12:00 am is Monday morning, not Sunday night. One minute late is late. Submitting Sunday night will allow the TA time to collect your submissions and gather them onto his laptop for evaluation. Because this is not a simple "hand it in" submission procedure, the requirement is also that every student be present at the lab/discussion section to explain their code to the TA.

Sometimes unexpected events make it difficult to get a project in on time. For this reason, each student will have a total of **3 free late days** to be used for projects throughout the semester. **These late days should only be used to deal with unexpected problems such as computer crashes, illness, etc.** They should not be used simply to start later on a project or because you are having difficulty with the project.

Projects received after the start of Monday morning's recitation section following the due date (assuming that you have no late days left) will receive a **zero**, even if you walk in the door one minute late. I advise you to save at least one or two late days for the last projects.

How Projects Are Graded

Again, you will explain your code to the TA, and you will demonstrate it during the Friday lab. The projects will be graded primarily for correctness: doing all the required tasks, adhering to whatever given time requirements are specified, and giving correct results. We will test your projects on various input sets that are not the same as what is handed out with the project.

Thoughts on Collaboration

Regarding what is and is not okay to do (thanks to the folks at Wisconsin) …

It is considered PERFECTLY ACCEPTABLE to do the following:

- discuss the project in general terms *("what do they mean by a vector table?")*
- discuss strategies for successful implementation *("our data-structure format is simple!")*
- help others debug small snippets of their code and find problems
- ask the TA or professor or both for as much help as you need!

It is NOT OK to do the following:

- bug someone else for a lot of help (particularly if they are already done!)
- share your code directly with other people *("oh, you want to know how to get the timer to stop interrupting? well here is my code, and it works, so you can just use/copy that")*

Discovery of any inappropriate code sharing will lead to harsh penalties for all involved parties. This draconian policy is put in place to protect the vast majority of you who ***do*** put in the hard work on the projects.

Exams

You are expected to take both the midterm and final exams at the scheduled times. Unless a (documented) medical or personal emergency is involved in your missing an exam, you will receive a zero for that exam. If you anticipate conflicts with the exam time, you must come talk to the instructor about it at least 1 month before the exam date. The exam dates are given at the beginning of the term so that you can avoid scheduling job interviews or other commitments on exam days. Outside commitments are not considered a valid reason for missing an exam. Exams will be closed book, closed notes.

Grading Policy

Final grades will be based on the total of points earned on the projects and exams. The tentative point breakdown is as follows. For those of you with mad math skills, yes, you get 2% extra credit.

- Projects: 52% (13 projects, 4% each)
- Midterm Exam: 25%
- Final Exam: 25%

Incompletes will generally not be given. According to university policy, doing poorly in a course is not a valid reason for an incomplete. If you are having problems in the course, your best bet is to come talk to the instructor as soon as you are aware of it.

Tentative Lecture & Project Schedule

Week of	Subject	Readings	Projects
Jan 25	Intro: overview of course, the concept of time & timing	Ch. 2, 3	P0 out
Feb 1	How hardware actually works (interrupts, I/O regs, call stacks, etc.)	Ch. 4–6, 33	P1 out
Feb 8	Interprocess Communication (IPC)	Ch. 25–27, 32, 47	P2 out, P0 due
Feb 15	Synchronization & deadlocks	Ch. 7–10, 28–31	P3 out, P1 due
Feb 22	Processes & scheduling & security	Ch. 11,12	P4 out, P2 due
Mar 1	Memory management & virtual memory — software	Ch. 13–18	P5 out, P3 due
Mar 8	Memory management & virtual memory — hardware	Ch. 19–24	P6 out, P4 due
Mar 15	***Spring Break***		
Mar 22	**Review & Midterm** (March 25, during class)		P7 out, P5 due
Mar 29	Persistence, permanent object stores	Ch. 35–37	P8 out, P6 due
Apr 5	File systems, data integrity	Ch. 38–41	P9 out, P7 due
Apr 12	Flash basics, ECC, etc.	Ch. 42–45	P10 out, P8 due
Apr 19	Advanced topics & case studies	Handouts	P11 out, P9 due
Apr 26	Advanced topics & case studies	Handouts	P12 out, P10 due
May 3	Advanced topics & case studies	Handouts	P11 due
May 10	**Final Review**		P12 due
Exams	**Final Exam** (Thursday, May 13, 8:00am–10:00am)		

Special Needs

If you have a documented disability that requires special needs, please see me as soon as possible, and certainly no later than the third week of classes.