ECE 8400 / ENGI 9875:
[Embedded and] Real-time [Operating] Systems [Design]

Lecture 0:

Welcome

http://localhost:8400/Engineering/ECE/Teaching/operating-systems/website/lecture/0/

Welcome!

ECE 8400: Real-time operating systems

ENGI 9875: Real-time and embedded system design

This lecture material supports two courses: one undergraduate and one graduate. These courses contain identical content, but differ in two respects:

1. the graduate course has higher expectations in terms of marking and
2. each graduate student must complete a self-directed project in addition to completing the more structured material used to evaluate undergraduate success.

Our goal

Learn to build engineered **systems** that use hardware and software components **effectively**

Relevant for:

- software developers
- hardware developers
- engineers designing other kinds of systems

The top-level goal for this course is not to turn you all into OS developers (although this material *is* a foundation for that ambition!). Rather, before we send you out into the world as Computer Engineers, we need to help you learn to build systems that include software that _____ _____. This may not seem relevant to your immediate goals, but it is important for more disciplines than you might think!

Operating systems for...

... software people?

```
PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(name)));
```

- why use `java.io.BufferedWriter`?
- how does `out.format(...)` actually work?
- how can we diagnose whole-system problems?

Without understanding how things actually work, we are doomed to the practice of *superstitious programming*: using incantations that seem to work without knowing if there are surprises in store, or if something else could work better. The reasons for using a `java.io.BufferedWriter` don't make a lot of sense in terms of the abstractions provided by Java: they only make sense when you consider the system as a whole (including the operating system). In this course we'll see how high-level I/O abstractions are written in terms of low-level C code, which interfaces with an OS kernel, which interfaces with hardware via device drivers; understanding the broad strokes of this stack is critical for understanding _____.

Operating systems for...

... hardware people?

- hardware isn't much good without software
- your shiny new architecture is useless without support from **compilers** and **operating systems**
- must build hardware that software can use
- "hardware-software co-design"

Hardware has lots of great properties, chief among them massive fine-grained parallelism, but that's not very helpful in the real world without _____. Where hardware is durable, software is _____.

If anyone is going to make use of your hardware, then, they will need to be able to write software for it. This requires compilers and - in all but the must trivial of cases - operating system support. It is possible to create a brand-new architecture in the 21st century, but not without an incredible amount of work from the compiler and OS communities.

Furthermore, it's not enough to just build hardware that people *could* write software for: you need to build hardware that software can exploit _____. This requires understanding something of what's happening on the other side of the ISA, and on hardware that's used to do _____, the main actor on the software side is the operating system.

It's both convenient and experimentally sound in a complex system to hold everything constant except for one component, then vary that component and measure the results. For example, if we can change some software while holding the hardware and OS constant, we have a nice, easy and _____ experiment. However, the most interesting things in life aren't that simple. Addressing hard problems requires work across many levels of abstraction: hardware, software and the things in between (compilers and operating systems). This is called "hardware-software co-design", and while it's fiendishly difficult, it's _____.

Operating systems for...

... other people?

Computer Engineers build:

- embedded systems that control larger systems
- computers with real-time safety-critical uses

Can you do this reliably?

Few engineered systems incorporate no aspect of software monitoring or control. Anybody can attach an Arduino to some sensors - our first-year programming course is moving in that direction! When computer engineers build an embedded system, there may be much more stringent requirements, including _____ implications.

To build a reliable embedded system, it's not enough to write some C++ and hope that it works. You need to understand the _____ and how it works _____, which requires an understanding of the software at the centre - the _____.

Our goal

Learn to build engineered **systems** that use hardware and software **effectively**, as shown in our **learning outcomes**:

- **explain** how operating systems mediate between applications and hardware
- **synthesize** correct low-level software in C
- **synthesize** OS primitives with concurrency
- **explain** the unique constraints of real-time and embedded systems and implications for their design
- **design** software to efficiently take advantage of OS primitives

How/when?

Time(s)	Occasion	Location
MTF @ 1pm	Lectures	EN-2040
Tue @ 1pm	Office hours	CSF-4123
Mon @ 2pm	Labs	CSF-2112

... plus your own hours

In addition to meeting for three lectures each week, we will also meet for five labs on Monday afternoons (using CSF-2112). We may also use some other Mondays for tutorial sessions.

Formal office hours for this course are Mondays from 1-2pm in my office (CSF-4123), but I also try to be as accessible where possible outside of these hours.

You should also plan to spend quality time working on assignments and prelabs. These will _____ the material and _____ to your future career goals (at least on a good day!).

Evaluation

Undergraduate

What	When	How much
Assignments (5)	Jan-Mar	20%
Labs (5)	Jan-Mar	20%
Midterm	17 Feb	20%
Final exam	???	40%

The evalution in this course is subject to the rule that _____ _____. That is, if you fail the weighted average of the midterm and the final, *that* is the mark you will receieve: you won't benefit from your assignments and labs.

Also, as this is a lab course, _____. Failure to complete all labs will lead to an _____, which may delay your final promotion decision.

Evaluation

Graduate

What	When	How much
Assignments (5)	Jan-Mar	20%
Labs (5)	Jan-Mar	20%
Project	24 Jan, 14 Feb, 28 Mar	10%
Midterm	17 Feb	15%
Final exam	???	35%

The evalution in this course is subject to the rule that _____ _____. That is, if you fail the weighted average of the midterm and the final, *that* is the mark you will receieve: you won't benefit from your assignments and labs.

Also, as this is a lab course, _____. Failure to complete all labs will lead to an _____, which may delay your final promotion decision.

Academic integrity

"I did this"

As a class of soon-to-graduate proto-Engineers and a class of mature graduate students who are responsible for the research integrity, I expect high standards academic integrity from you.

When I ask you to do individual work, I expect that _____. When you turn your work in, you should be able to say honestly, "I did this". Others (including myself!) may have helped you understand the ideas, understand the problem, understand the question, etc., but the work _____.

Resources

No traditional textbook

Operating Systems: Three Easy Pieces (http://ostep.org) : Free online book that grew out of an OS course at the University of Wisconsin

Other electronic resources (e.g., manuals)

Labs

Practical learning experiences:

- programming in C
- communicating with the kernel
- low-level threading
- VM paging
- bootloading

Some prelabs will be _____

Course website

https://memorialu.gitlab.io/Engineering/ECE/Teaching/operating-systems/website

Documents

- the course outline (read it!)
- reference manuals for lab hardware
- architecture and ABI references

News (e.g., weather!), Lectures (and notes)

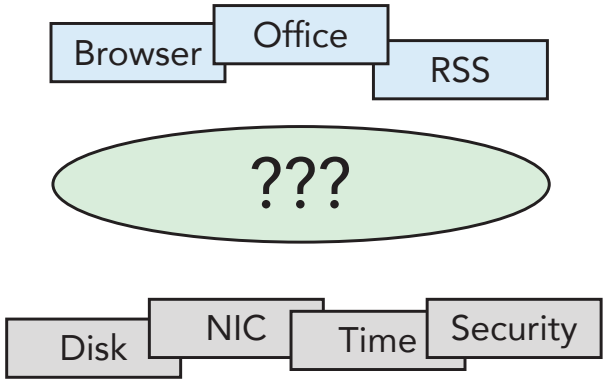
In addition to the lecture slides, I will try to include notes where I have something to add beyond what others have written (in OSTEP or elsewhere). If I don't have much to add, the lecture notes will at least make reference to the appropriate reference materials. Either way, I'll also include links to relevant code snippets where appropriate.

What do operating systems do?

- Process execution, scheduling
- Inter-process communication
- Hardware I/O
- Hardware abstraction
- Filesystems
- Access control (system, files, memory, devices...)
- Accounting

Of course, there's lots of debate about what an operating system does: these are a few categories that are fairly uncontroversial. Although, the last category could cover a lot of ground...

What do operating systems do?



Operating systems _____ software access to _____, whether concrete (disk space, I/O devices) or abstract (time, security).

Real-time operating systems

Why are "real-time" systems so special?

- predictable
- priority
- periodic (often)

Hard real-time ("or else"): cars, avionics, Mars robots...
Soft real-time ("mostly"): A/V, transactions, simulation...
Non-real-time: everything else!

Real-time systems don't necessarily have to be faster than other systems, but their behaviour must be more _____ than other systems. There is some notion of _____ that must be met (at least stochastically), usually with clear notions of the relative priorities of various _____. These tasks are often (but not always) _____ (e.g., "update the oxygen sensor reading and its first derivative every 5ms").

A _____ system has _____, often with emphasis on the "dead". If a flight control surface fails to react to pilot input in a timely way, it can result in a number of people having a rather bad day.

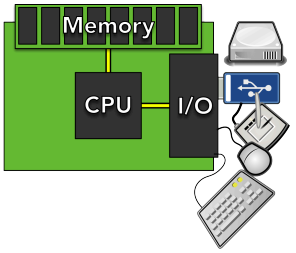
A _____ system has _____, in which it's important for deadlines to be met _____, but some slippage is acceptable. For example, there may be quality-of-service guarantees about frames being dropped in a high-quality video stream, or a server might need to be able to service at least N queries per second with high confidence.

Everything else (including the computer that this lecture is presented from) falls into the category of *non-real-time* systems.

Towards the end of the course we'll explore how real-time systems are implemented in terms of task timing constraints, scheduling, priority mechanisms (including prevention of priority inversion) and preemption blocking.

Things to remember

From first-year programming:



- Simplistic model of a computer
- Three parts:
  - CPU executes instructions
  - Memory stores data
  - I/O devices interact with the outside world

This is the same picture that I show students in ENGI 1020: Introduction to Programming. The essence of a computer is a CPU (to _____) connected to memory (for _____) and I/O devices (for _____).

More things to remember

From Microprocessors and Computer Architecture:

- Interrupts
- Memory and I/O interfacing
- Multiprocessing
- Privilege modes
- Virtual memory

These concepts ought to _____. If not, you may need to _____ _____.

Virtual memory

- Virtual and physical addresses
- Page tables and TLBs

Next time...

History of operating systems