Introduction to ECE 391

Michael Bailey
University of Illinois at Urbana-Champaign

Fall 2021

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Office of the Provost

Dear Colleagues,

We have shared several updates about Fall 2021 in the past several weeks as we monitor the progress with managing the pandemic and respond to new science-based COVID-19 guidance from the CDC, IDPH, CUPHD and our own SHIELD team. Below is a condensed version of what we need to know as we prepare for in-person classes in a few weeks.

Face Coverings

- Everyone (faculty, staff, students, visitors) is required to wear a face covering in university facilities. Read more about face coverings and face shields here.
- If a student is not wearing a face covering in your class, ask them to put one on. If they refuse, dismiss the class and report the student to the Office for Student Conflict Resolution for further discipline by filling out this form. Call UIPD, 217-333-1216, only if an individual becomes belligerent, disruptive and threatening.

Welcome to ECE391!

- Today: Course Introduction
  - Introductions
  - Course philosophy
  - Course logistics
  - Advice from alumni

INTRODUCTIONS

Who We Are

- Instructors:
  - Yih-Chun Hu
  - Michael Bailey
- Teaching Assistants:
  - Haotian Chen
  - Chenyang Huang
  - Sung Woo Jeon
  - Xiang Li
  - Yan Miao
- Lab assistants

COURSE GOALS

Computer Systems Engineering

"Concepts and abstractions central to the development of modern computing systems, with an emphasis on the systems software that controls interaction between devices and other hardware and application programs."

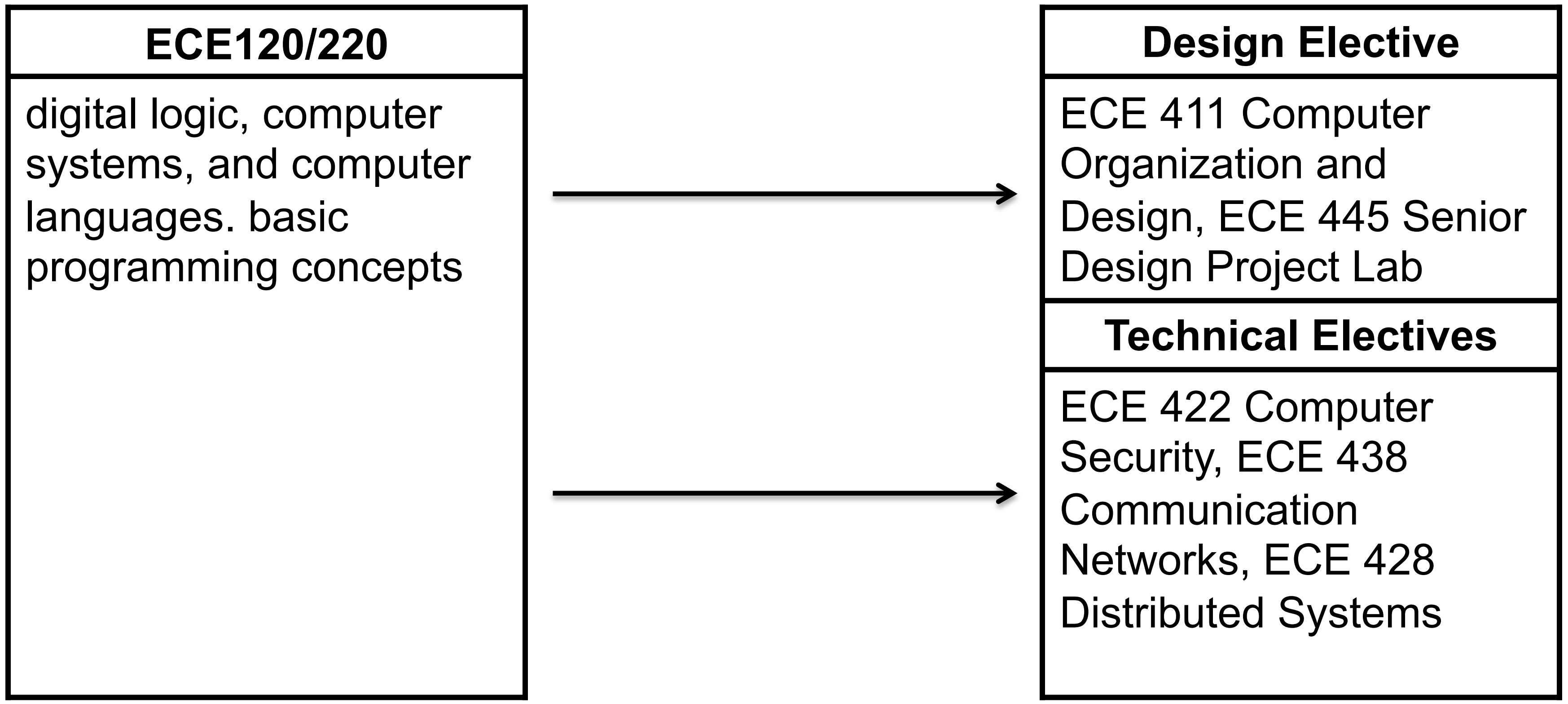
Computer Systems Engineering

"Concepts and abstractions central to the development of modern computing systems, with an emphasis on the systems ***software that controls interaction between devices and*** *other hardware and* ***application programs****."*

Course Goals

ECE120/220
digital logic, computer systems, and computer languages. basic programming concepts

Course Goals


ECE120/220
digital logic, computer systems, and computer languages. basic programming concepts
Design Elective
ECE 411 Computer Organization and Design, ECE 445 Senior Design Project Lab
Technical Electives
ECE 422 Computer Security, ECE 438 Communication Networks, ECE 428 Distributed Systems

Course Goals

ECE120/220	ECE391	Design Elective
digital logic, computer systems, and computer languages. basic programming concepts	Machine-level operations, system software, virtualization of resources, interrupts, data movement	ECE 411 Computer Organization and Design, ECE 445 Senior Design Project Lab
		Technical Electives
		ECE 422 Computer Security, ECE 438 Communication Networks, ECE 428 Distributed Systems

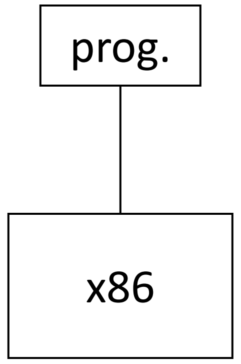
ECE391 bridges the gap between your intro classes and design and technical electives courses through system-level programming of a real (x86) computer

Bridging the Gap

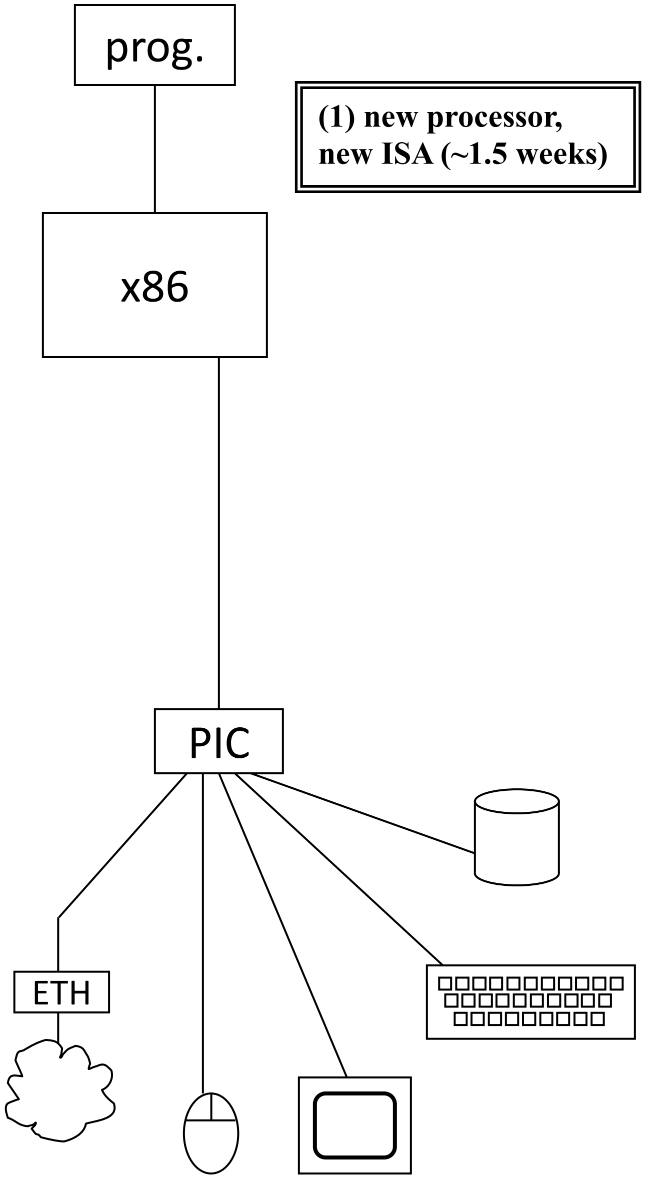
- review of **computer organization** and representations
- **x86 assembly**: review of basic constructs and structures, interfacing C to assembly, macros, stack frame and calling convention
- simple **data structures**: queues, heaps, stacks, lists
- **synchronization**: primitives, memory semantics, mutual exclusion, semaphores, scheduling, and race conditions
- **interrupts**: controlling generation and handling, chaining, cleanup code, interactions with device functionality
- **resource management**: virtualization and protection, virtual memory and hardware support

Bridging the Gap

- **exceptions and signals**: exceptions due to instructions, memory accesses, and floating-point operations; signal semantics and delivery
- **device programming:** basic abstractions, character and block devices
- **file system abstractions**: disk layout, access control lists and capabilities, log-structured and traditional filesystems as design problem
- **I/O interface:** file descriptors, buffering, control operations, memory mapping
- **networking programming:** socket abstractions, basics of low-level network protocols, relationship to kernel I/O abstractions



(1) new processor, new ISA (~1.5 weeks)



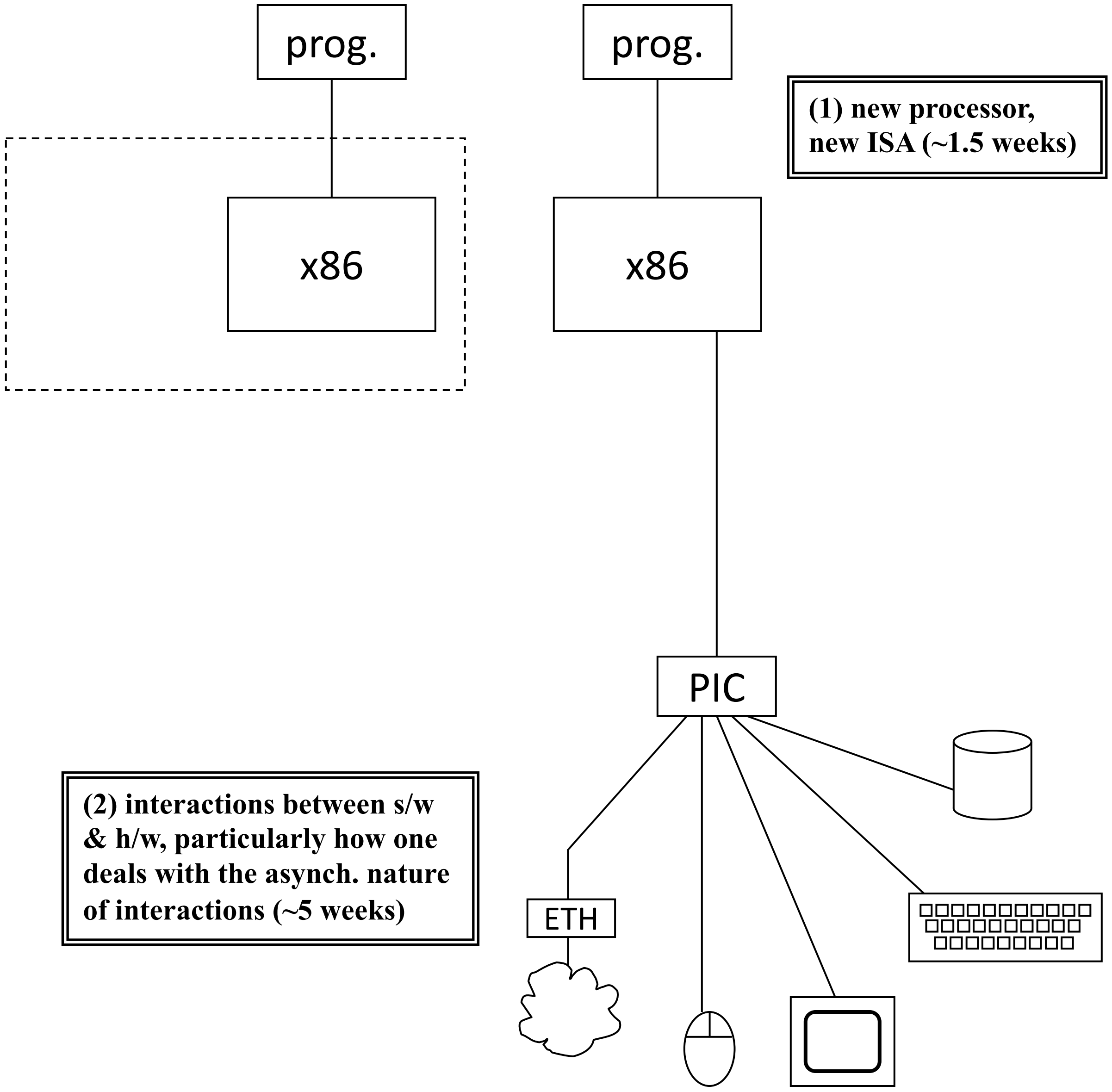
(1) new processor, new ISA (~1.5 weeks)

(2) interactions between s/w & h/w, particularly how one deals with the asynch. nature of interactions (~5 weeks)

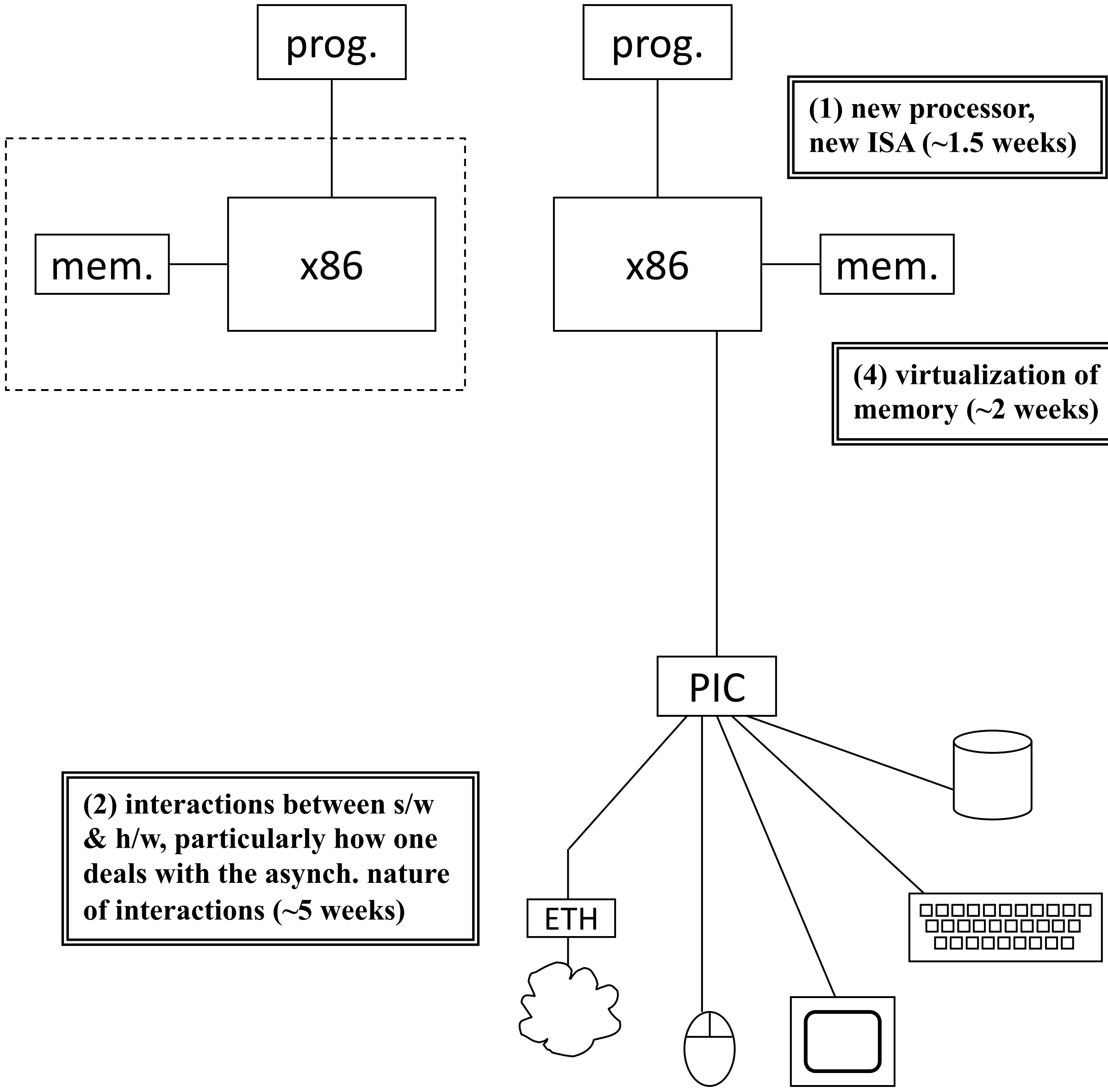
(3) virtualization (starting with the processor): providing the illusion of a private machine (~1.5 weeks)

(1) new processor, new ISA (~1.5 weeks)

(2) interactions between s/w & h/w, particularly how one deals with the asynch. nature of interactions (~5 weeks)



(3) virtualization (starting with the processor): providing the illusion of a private machine (~1.5 weeks)

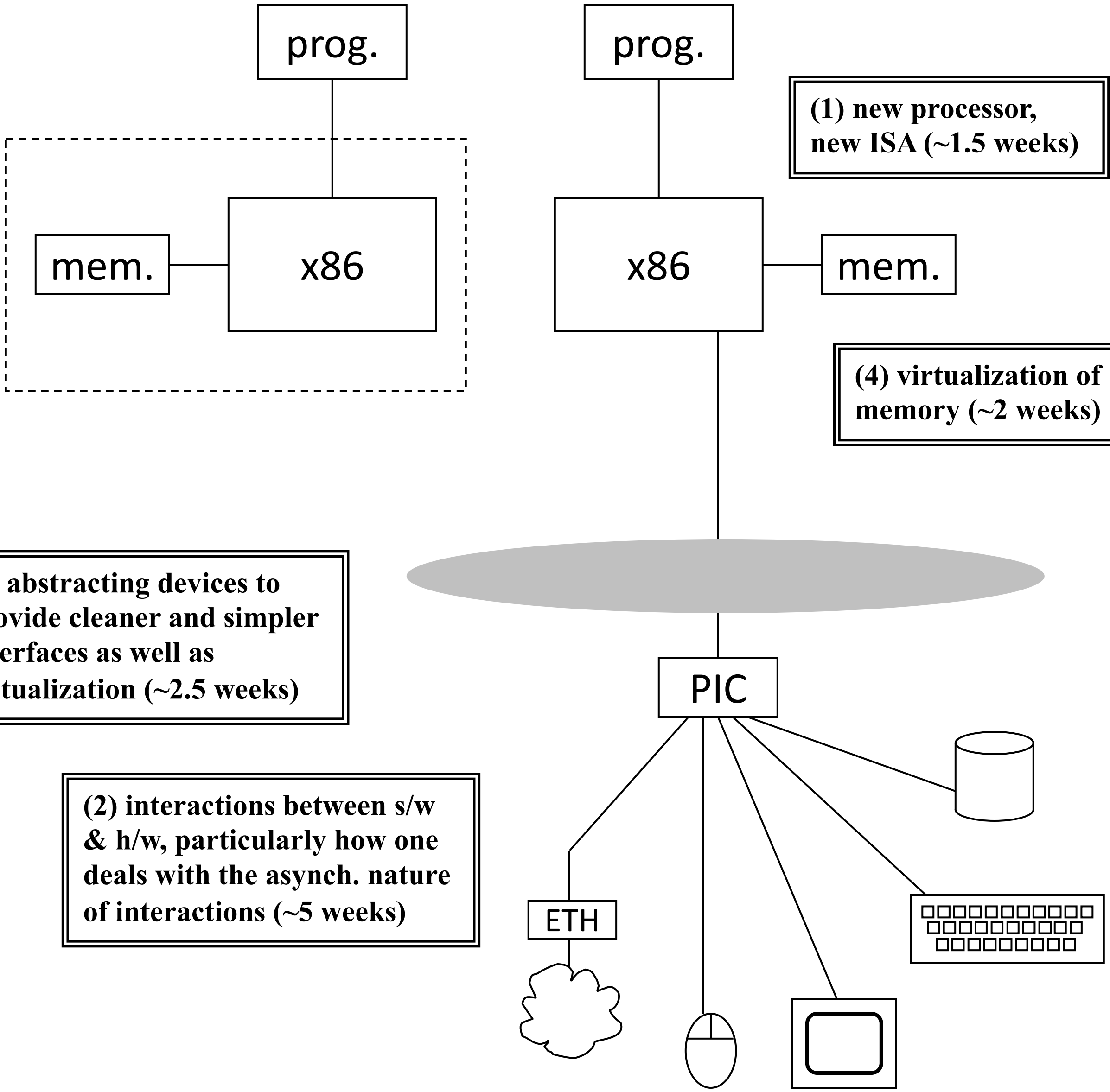


(1) new processor, new ISA (~1.5 weeks)

(4) virtualization of memory (~2 weeks)

(2) interactions between s/w & h/w, particularly how one deals with the asynch. nature of interactions (~5 weeks)

(3) virtualization (starting with the processor): providing the illusion of a private machine (~1.5 weeks)



(1) new processor, new ISA (~1.5 weeks)

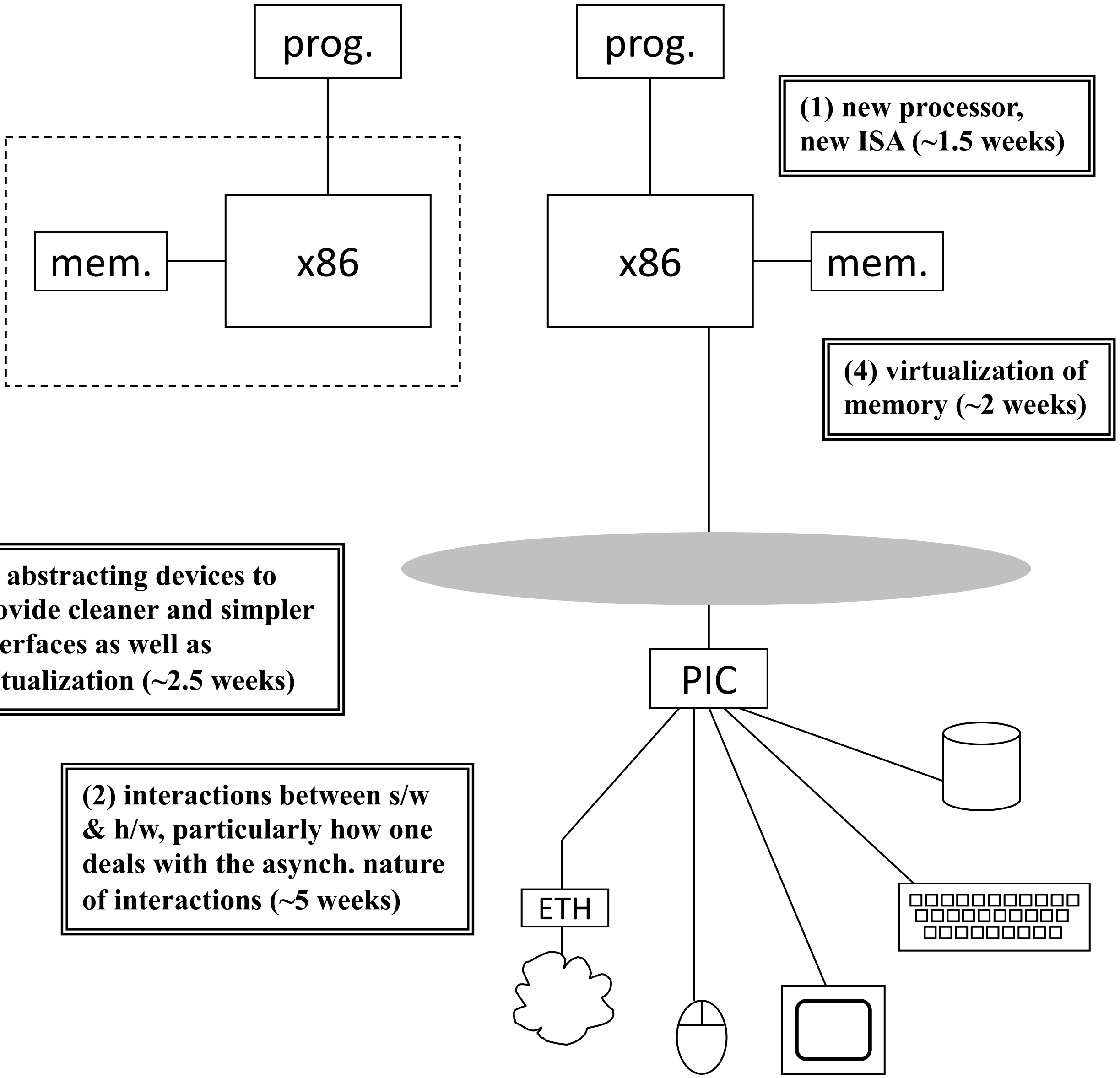
(4) virtualization of memory (~2 weeks)

(5) abstracting devices to provide cleaner and simpler interfaces as well as virtualization (~2.5 weeks)

(2) interactions between s/w & h/w, particularly how one deals with the asynch. nature of interactions (~5 weeks)

(6) user-level interrupts, or signals (~1 week)

(3) virtualization (starting with the processor): providing the illusion of a private machine (~1.5 weeks)



(1) new processor, new ISA (~1.5 weeks)

(4) virtualization of memory (~2 weeks)

(5) abstracting devices to provide cleaner and simpler interfaces as well as virtualization (~2.5 weeks)

(2) interactions between s/w & h/w, particularly how one deals with the asynch. nature of interactions (~5 weeks)

(3) virtualization (starting with the processor): providing the illusion of a private machine (~1.5 weeks)

(6) user-level interrupts, or signals (~1 week)

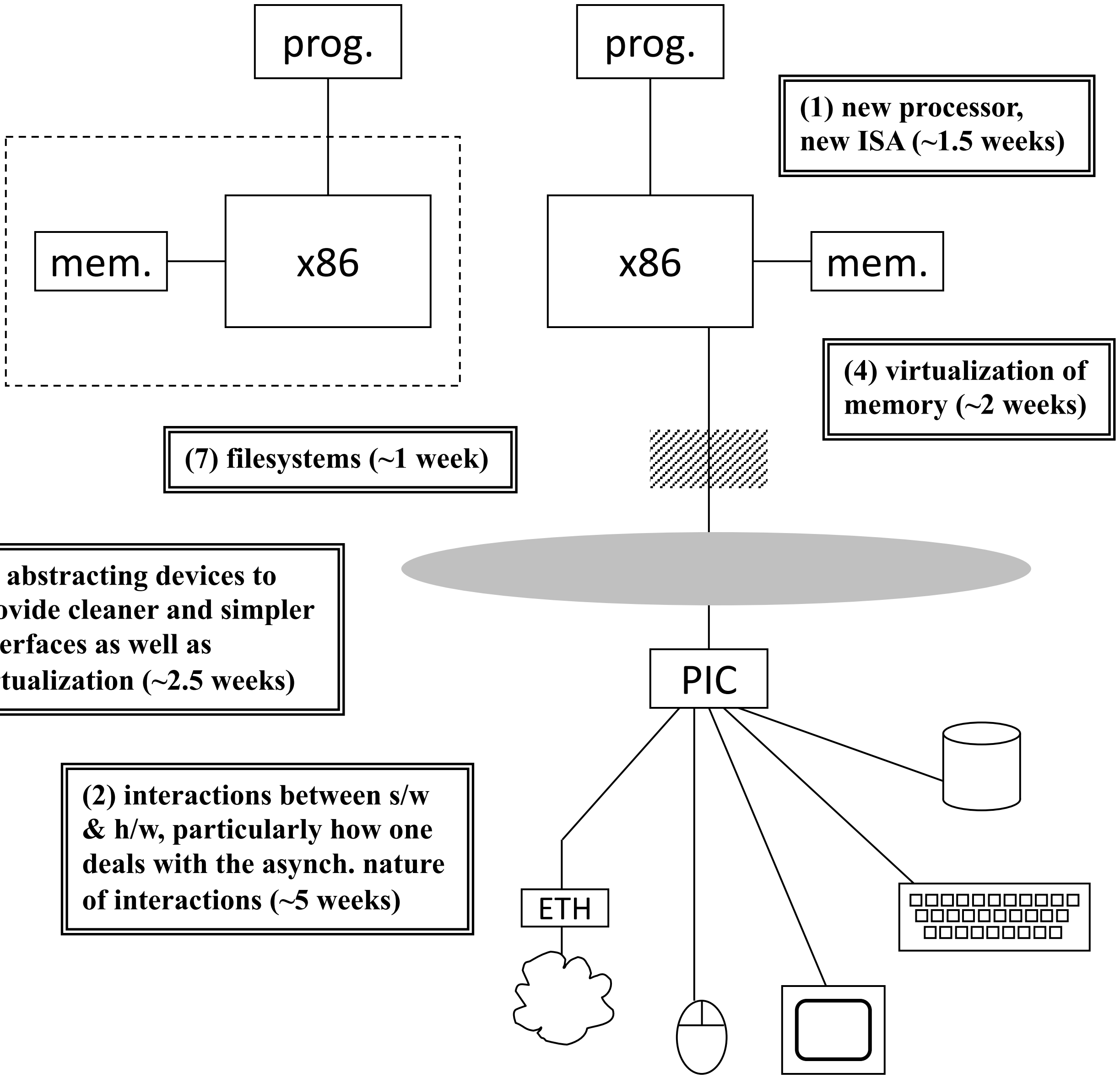
(1) new processor, new ISA (~1.5 weeks)

(4) virtualization of memory (~2 weeks)

(7) filesystems (~1 week)

(5) abstracting devices to provide cleaner and simpler interfaces as well as virtualization (~2.5 weeks)

(2) interactions between s/w & h/w, particularly how one deals with the asynch. nature of interactions (~5 weeks)



The “Other Gap”

- MP1: about 20 instructions in LC-3 binary (written as bits, in one checkpoint)
- MP2: around 250 lines of LC-3 assembly (done as two checkpoints)
- MP3: around 100 lines of simple C (one checkpoint)
- MP4: 1000 lines of simple C (a big switch statement for an LC-3 assembler, done as two checkpoints)
- MP5: 300 lines of more complex C (done as two checkpoints)

- MP1: 300 lines of x86 assembly
- MP2: add 500 lines to an existing code of 4-5k lines (two checkpoints)
- MP3: team of 4 write an operating system, usually around 3k lines or so

The “Other Gap”

Snippets from ECE 422 FAQ

- This is an intensive, systems-orientated class with considerable time required to complete the course. I expect each of the five machine problems (MPs) to take roughly 20 hours each.
- Do not take this class if you are uncomfortable with significant *independent* inquiry. If your education to this point did not include, for example stacks, virtual memory, or networking concepts, we expect you will fill in these gaps yourself.
- We anticipate you have learned (n) unique programming languages by now and that learning a new one is trivial; we will ask you to learn several new ones without assistance.
- We expect you have built *lots* of computing artifacts, that you like to build them, and are already familiar with reasoning about and designing for other systems properties such as performance and correctness. We will not give you access to the auto graders and we will not test your code for you.

The “Other Gap”

- Real systems are built using real tools
- You will learn something about using …
  - source control
  - compilers
  - dependency management (make files)
  - debuggers

THE TRUTH BEHIND ECE 391

8

Posted by u/Waifu4Laifu ECE '17 7 years ago

The Truth behind ECE 391?

I'm currently a EE and I've been thinking about switching to CE and taking ECE 391. From what I see on Facebook and here is that it is one of (if not THE) hardest class at the U of I.

No doubt it'll be really tough, but how much of a timesink is it really?

Am I fucked if I don't get good teammates/am a Linus Torvalds 2.0?

Will I be spending weeks at a time in the labs in a sleeping bag eating packaged food and drinking red bull?

I'm not sure whats myth and whats truth at this point.

I just started programming in college and I'm not sure if taking it is a good idea due to my inexperience. I've taken 198jl and am taking KL, and will be taking 225 next semester. I really love programming and 391 seems like a class that will exponentially increase my understanding.

If I dont take it, I'll probably take CS241 instead and getting a CS minor.

Thanks for any input!

31 Comments Share Save Hide Report 85% Upvoted

COURSE PHILOSOPHY

Neil Armstrong

- "…I had a physics professor who had written our textbook and for the first Friday recitation I anticipated that I would need to regurgitate the assigned chapter. Instead, the professor said, 'I'm curious what you *thought* about this material.' At that moment I realized … it was about teaching problem-solving, critical thinking, analyzing situations, and coming to conclusions that were in detail and original."

James R. Hansen, "First Man: The Life of Neil A. Armstrong," Simon & Schuster, New York, New York, 2005.

Neil Armstrong

- “…I had a physics professor who had written our textbook and for the first Friday recitation I anticipated that I would need to regurgitate the assigned chapter. Instead, the professor said, 'I'm curious what you *thought* about this material.' At that moment I realized … **it was about teaching problem-solving, critical thinking, analyzing situations, and coming to conclusions that were in detail and original**.”

James R. Hansen, "First Man: The Life of Neil A. Armstrong," Simon & Schuster, New York, New York, 2005.

Up to now we taught you programming …

- A lot of students when they graduate and get their first job, feel like they don't really know how to program even though they may have been good programmers in college.

https://softwareengineering.stackexchange.com/questions/119470/differences-between-programming-in-school-vs-programming-in-industry

Up to now we taught you programming …

- A lot of students when they graduate and get their first job, feel like they don't really know how to program even though they may have been good programmers in college.
- What are some of the differences between programming in an academic setting and programming in the 'real world'?

https://softwareengineering.stackexchange.com/questions/119470/differences-between-programming-in-school-vs-programming-in-industry

Up to now we taught you programming …

- A lot of students when they graduate and get their first job, feel like they don't really know how to program even though they may have been good programmers in college.
- What are some of the differences between programming in an academic setting and programming in the 'real world'?
- **…the real world doesn't want people who are just programmers**

https://softwareengineering.stackexchange.com/questions/119470/differences-between-programming-in-school-vs-programming-in-industry

Systems is more than just programming

- Gather and analyze requirements when they aren't directly given to you
- Design and analyze architectures with near endless possibilities
- Create test plans and act on them to evaluate and improve the quality of a system

https://softwareengineering.stackexchange.com/questions/119470/differences-between-programming-in-school-vs-programming-in-industry

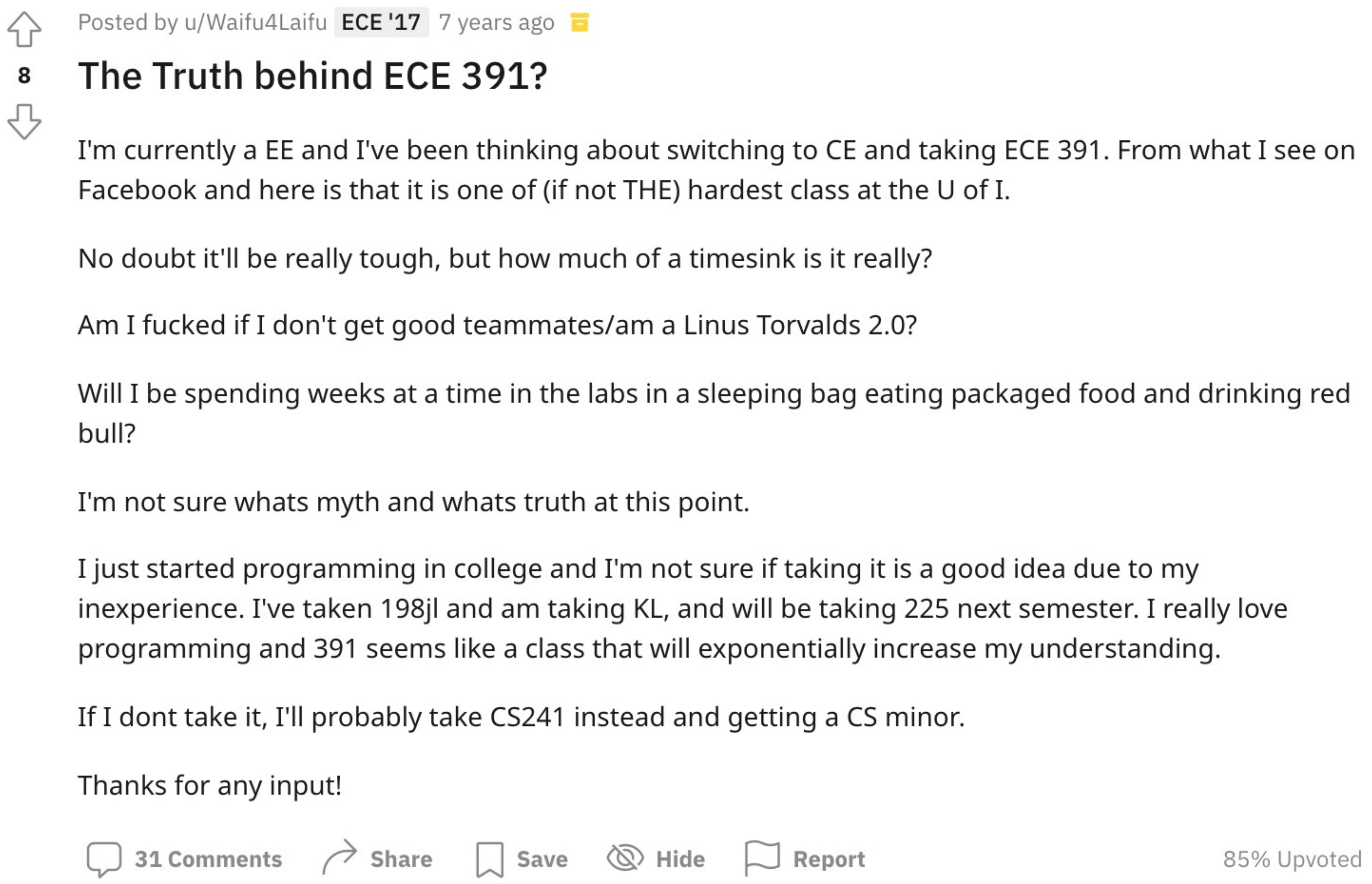
Systems is more than just programming

- Messy and large codebases
- Learning to READ code
- Work collaboratively on a team of people with different backgrounds and experience levels
- Estimate and plan work even if you don't know exactly what to build

https://softwareengineering.stackexchange.com/questions/119470/differences-between-programming-in-school-vs-programming-in-industry

Listen to ECE graduates

DON’T BELIEVE US?



8

Posted by u/Waifu4Laifu ECE '17 7 years ago

The Truth behind ECE 391?

I'm currently a EE and I've been thinking about switching to CE and taking ECE 391. From what I see on Facebook and here is that it is one of (if not THE) hardest class at the U of I.

No doubt it'll be really tough, but how much of a timesink is it really?

Am I fucked if I don't get good teammates/am a Linus Torvalds 2.0?

Will I be spending weeks at a time in the labs in a sleeping bag eating packaged food and drinking red bull?

I'm not sure whats myth and whats truth at this point.

I just started programming in college and I'm not sure if taking it is a good idea due to my inexperience. I've taken 198jl and am taking KL, and will be taking 225 next semester. I really love programming and 391 seems like a class that will exponentially increase my understanding.

If I dont take it, I'll probably take CS241 instead and getting a CS minor.

Thanks for any input!

Systems Thinking

- 391 benefited me greatly in my career … Even with architectural differences of working on Windows, 391 helped teach me how to approach and solve system-level problems.

- Drew Kluemke, ECE 391 in Fall 2015, Software Engineer at Microsoft

Systems Thinking

- Computer systems and cars are similar machines: in the sense that you don't have to know how to build a car in order to drive or fix one, nor will it make you an expert driver … but if you understand the fundamental physics you will never be surprised with something you can't understand or learn.

- Daniel Fernandes, ECE in Fall of '13, Software Engineer at Cadence Design Systems

LMGTFY

- …the fundamental ideas taught in lecture and the hours reading dense technical documentation (Intel Manual) made it easy to learn quickly from the corresponding ARM manuals and Apple documents.

- Eric Clark, ECE 391 in Fall of '13, CPU Debug Engineer at Apple

Coding in Practice

- In my professional career, I have found that much of my time is spent reading new code and figuring out how to implement new features that that my org needs; these skills have transferred well from ECE 391.

- Matt Tischer, ECE 391 in Spring '12, Software Engineer at Microsoft

Real systems are messy

- In my job today … There are no instructions, no hints, no TA office hours, no nothing. It’s just me and hours of banging my head against a wall. But I’d like to think that 391 prepared me well and so I suffer from much less head-banging than I likely would have otherwise.

- Eric Carl of House Badger, the First of His Name, The Unallocated, King of the Stackman, the Bitdiddle and the First Ben, King of Execute, Kerneleesi of the Great Language C, Protector of the Stack, Lord Regnant of the Seven Syscalls, Breaker of Segments and Father of Pages, ECE in Spring 2012, Hadoop Code Monkey at Oath: the company formerly known as Yahoo!

Not any code, the right code

- There were so many things I learnt in ECE 391, but the most important one is how to write good code and why do we need to do so.

- Fei Deng, ECE 391 in Spring '14, Software Development Engineer at Oath

Printf?

- In ECE 391 you learn to be very methodical in how you debug and write code … I think this skill, perhaps more than any other, will impress your coworkers and bring value. I've had cases where people asked me "how did you find that bug?" when the answer was simply I opened the debugger and stepped through the code.

- Dennis Liang, Spring ‘14, Systems Engineer at Cloudflare

L2 Play Well With Others

- ECE 391 gave me experience in working on a large coding project within a group… The course also taught me how to better collaborate with group members and divide work in ways so group/team members don't "step on each other's toes."

- Michael F., ECE 391 in Fall 2015, now Software Engineer at Microsoft.

LOGISTICS

Course Components

- Lectures
  - Focus is on concepts and case studies
- Discussion sections
  - Help with lecture concepts and labs
- Pre-labs
  - Prepare you for the labs
- Labs
  - Experience with real systems
- Exams (2 midterms + 1 final)
  - Cover **both** lecture and lab material

Textbooks / Other Resources

- Course notes (online)
- Other books:
  - *Understanding the Linux Kernel (Bovet & Cesati, 3rd ed)*
  - *Linux Device Drivers (Corbet et al., 3rd ed)*
  - *Linux Kernel Development (Love, 3rd ed)*
  - *Design of the UNIX Operating System (Bach)*
  - *Advanced Unix Programming (Rochkind, 2nd ed)*
  - *Unix Systems Programming (Robbins & Robbins)*

Getting Help

- Piazza
  - Stop here first!
  - All announcements & clarifications posted here
  - Can ask public and private questions
  - Limited anonymity
- Office hours
  - Schedule will be posted this week
- Email
  - Last resort!

Grading

- MPs: 50%
  - MP0: 5%, MP1: 10%, MP2: 10%, MP3: 25%
- Prelabs: worth 10% of MP grade
- Exams: 15% each
- Other: 5% subjective evaluation

Collaboration

- Pre-labs must be done in groups
  - One hand-in per group is allowed
  - Must be typed
- MP 0, 1, 2 must be done individually
- MP3 must be done in a team of four
  - Start thinking about partners now!
- Any unauthorized collaboration on MPs or exams is a violation of academic integrity

Lab

- 3026 ECEB
  - 44 Windows workstations
- Reserved 24/7 for this class only
- Other EWS workstations
  - 2022 ECEB, Engineering Hall, Grainger, and Mech Eng Lab

Lab Rules

- There *will* be contention for lab resources
  - Start early!
- For MP 0, you may lock your computer for up to *4 hours*:
  - Write a note with time you will be back
- After MP 0, lock limit is 30 minutes
- Hand-in/demo takes precedence
- Working from home allowed (& encouraged) but demos must happen on lab machines
- Be respectful

Online Students

- Lectures recordings via Echo 360
- One of the two ONL discussion sections via Zoom
- Office hours via specially marked ONL office hours via Queue and Discord
- Remote access to Windows workstations by request

What is an infraction of academic integrity?

- **Cheating** – using or attempting to use unauthorized materials
- **Plagiarism** – representing the words, work, or ideas of another as your own
- **Fabrication** – the falsification or invention of any information, including citations
- **Facilitating Infractions of Academic Integrity** – helping or attempting to help another commit an infraction
- **Bribes, Favors, and Threats** – actions intended to affect a grade or evaluation
- **Academic Interference** – tampering, altering or destroying educational material or depriving someone else of access to that material

https://provost.illinois.edu/policies/policies/academic-integrity/students-quick-reference-guide-to-academic-integrity/

Criteria and Sanctions

- Our criteria is "**more probably true than not true**", that you have committed an infraction; not "beyond a reasonable doubt."
- **ALL infractions will be reported** through FAIR (Faculty Academic Integrity Report)
- The most common sanction will be to **fail students from the course** (category 3 sanctions) for academic integrity violations.

Why the lecture?

- Pressure
  - Fear of Failure/Pressure to Succeed
  - Financial Support Scholarships, Family
  - Lack of time
- Rationalization
  - Awareness of what constitutes cheating
  - Perceived insignificance of behavior
  - Sense of justice
  - Peer behaviors
- Opportunity
  - Technology
  - Absence of in-class deterrents
  - Lack of faculty resources

- Pressure
  - This is in your control, heed our advice about how to succeed
- Rationalization
  - Raise awareness, demonstrate significance, set tone for community
- Opportunity
  - We actively deter and police, are well resourced, and have specialize tools

More words of wisdom from those who went before …

TIPS FOR SUCCESS

Time management

- **Starting early** is the most obvious but also most helpful piece of advice
- My advice to ECE 391 students is to **start early** on assignments and stay ahead of deadlines.
- Everything will **take longer than you think**
- **Plan ahead**. This sounds so simple, but it's actually the hardest thing to do
- **Time management in ECE 391 is critical**. While it may be possible to complete MPs in a single night in some other classes, ECE 391 is not among them.

Right Attitude

- I think ECE 391 can be a bit of a grind for some students because of its difficulty. However, consider this: **it's *worth* learning because it's difficult**. Learning doesn't stop when you're done with ECE 391…
- It is an important class, and a difficult one for many students, but **you don’t have to be a genius to do well**. The same hard work and dedication that got you here will carry you through this class, too.

L2Read

- My advice is, and has always been, **RTDC (Read the Documentation Carefully)**. There were so many times students ask questions that can be answered with a direct quote from the docs we gave out.

Don be “That Guy”

- If you let your teammates do everything, then you learn nothing and the exams are impossible. Don’t be that person. **Help your team and learn the stuff**.
- If I were to retake ECE 391, I would **spend more time getting involved with my teammates** ... casual code reviews and standups … would have ensured that everyone was staying up to date on the assignments, allowed bugs to be caught earlier, and keep everyone familiar with the entire.

The TAs Don’t Suck

- Go to discussions, pay attention to the TA since they've all done well in ECE 391. **Go to office hours, ask good questions** but don't expect the TA to help you debug.
- Talk to the TAs and your classmates. **Make sure you really understand what the class is asking you to do and why** you're doing it.
- **Make use of office hours**. The 391 staff is relatively large and hosts frequent office hours in the lab. Maximize the value of the questions you ask;

#define DEBUG 1

- **Get comfortable with your debugging setup-- you'll be using it a lot.**
  - Learn to use GDB, don't rely on "printf" to debug, that's what newbies do.
- Breaking down large tasks into smaller segments of code and **testing** each segment **early and often** really makes debugging easier as there are less items that can go wrong at each step.

l2code ≠ l2think

- Additionally**, try to think about the *best* way to solve a problem**. Lots of groups would hack something together to clear a checkpoint then never go back and clean up their technical debt..
- spend more time commenting your code and removing magic numbers, and spend **more time thinking about the architecture** of your OS design.
- Don't just try the first approach that comes to mind. **Convince yourself of the best one.** Be skeptical of online resources like Stack Overflow and OSDev Wiki.

Introduction to ECE 391

Michael Bailey
University of Illinois at Urbana-Champaign

Fall 2021