ECE391
Computer System Engineering

Lecture 1

Zbigniew Kalbarczyk

University of Illinois at Urbana- Champaign

Fall 2018

Welcome to ECE391!

- Today: Course Introduction
  - Staff intro
  - Course philosophy
  - Course logistics
  - Advice from alumni

Who We Are

- Instructors:
  - Prof. Zbigniew Kalbarczyk
  - Prof. Jian Huang
- TAs
  - Tianxi Zhao
  - Key-whan Chung
  - Huiyuan Liu
  - Peter Du
  - Andrew Smith
- Lab assistants

COURSE PHILOSOPHY

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	ECE391	Design Elective and Technical 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
		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

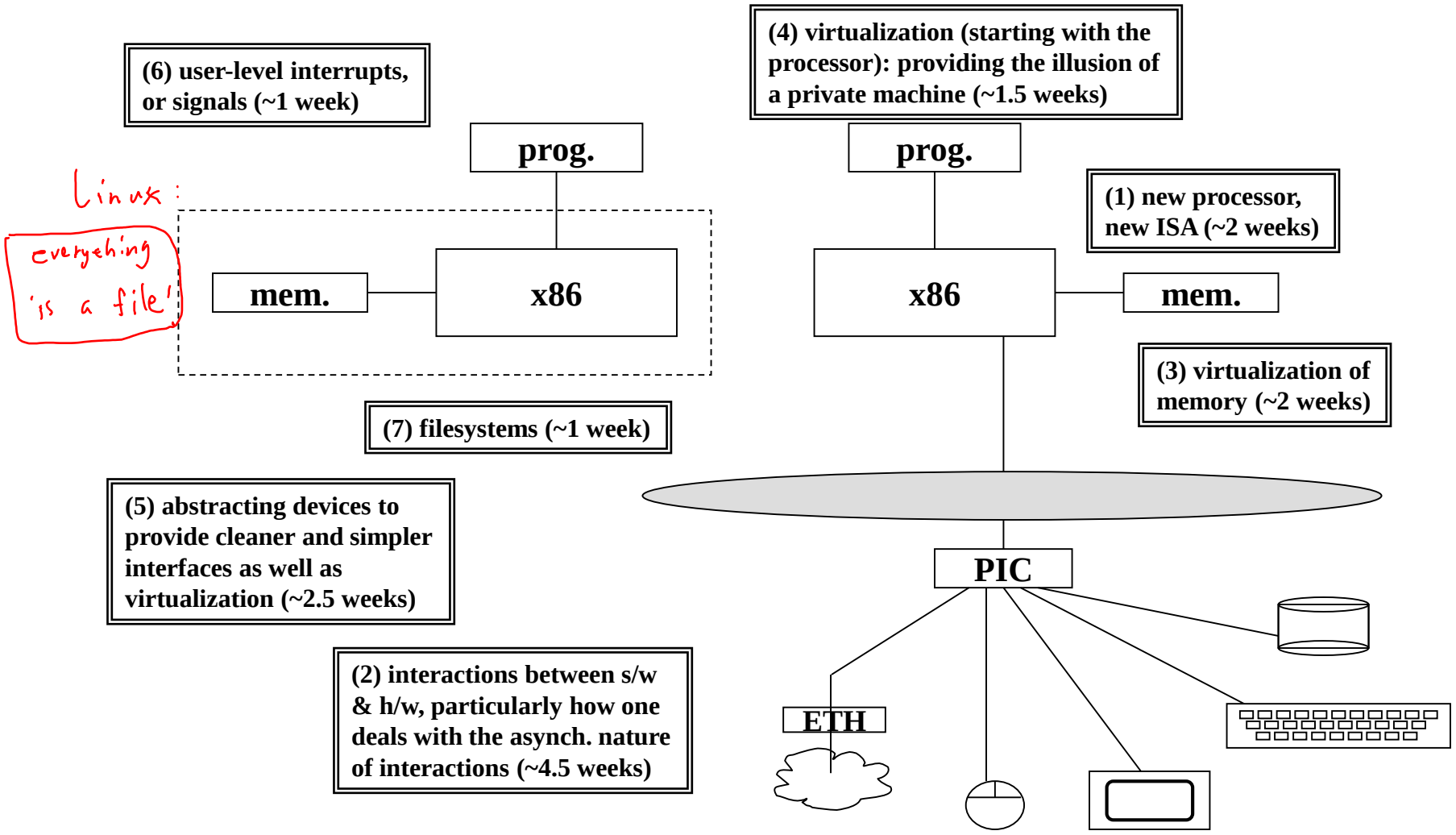
Topics

- 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

Topics (cont.)

- **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

Big Picture


(6) user-level interrupts, or signals (~1 week)
prog.
Linux:
everything 'is a file'
mem.
x86
(4) virtualization (starting with the processor): providing the illusion of a private machine (~1.5 weeks)
prog.
(1) new processor, new ISA (~2 weeks)
x86
mem.
(3) 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)
PIC
(2) interactions between s/w & h/w, particularly how one deals with the asynch. nature of interactions (~4.5 weeks)
ETH

Systems is more than just programming

- Gather and analyze requirements when they aren't directly given to you
- Design and analyze architecture 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

LMGTFY

- ECE 391 provided the foundation necessary to be succeed in my CPU design verification internship at Apple. Even though I wrote ARM assembly, 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, now ECE grad student.

Coding in Practice

- However, the value of the ECE 391 to me extends beyond memorizing particular concepts; it was in ECE 391 that I got my first real experience working in unfamiliar codebases and debugging challenging problems. **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, Spring '12, Software Engineer at Microsoft

Real systems are messy

- ECE 391 is pretty much the only class in college that challenged my coding skills. ECE 190 … and CS 225 gave me some exposure to coding, but at that point I was still learning. 391 was the first time that I got to actually work on a large code. **In my job today, I don't have the luxury of the answers already being figured out… 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, Spring 2012, Hadoop Code Monkey at Oath: the company formerly known as Yahoo!

Writing good 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**. ECE 391 prepared me really well for my job, aside from the point mentioned above, it also taught me how to think as an computer system engineer.

- Fei Deng, Spring'14, Software Development Engineer at Yahoo! Inc.

Systems Thinking II

- 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**. I also spend a significant amount of time debugging crash dumps, so the debugging experience from the MPs was very helpful, as well as being able to look at disassembly with the knowledge of x86 and calling conventions.

- Drew Kluemke, Fall 2015, 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 or technician (software engineering and I.T. in the computer world), but **if you understand the fundamental physics you will never be surprised with something you can't understand or learn**.

- Daniel Fernandes, Fall of '13, EE PhD Student at Stanford University

Debugging

- I think taking ECE 391 has earned me an enormous amount of respect in the workplace … one other thing I think is hugely important in ECE 391 is that 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

Groups

- 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., 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

Student Code

- Article 1. Student Rights And Responsibilities
- Part 4. Academic Integrity and Procedure
  - 1-401 Policy Statement; Application; Definitions
  - 1-402 Academic Integrity Infractions
  - 1-403 Procedures
  - 1-404 Sanctions
  - 1-405 Reporting and Recording Keeping
  - 1-406 Continuing Jurisdiction of the Senate Committee on Student Discipline

http://admin.illinois.edu/policy/code/article1_part4_1-401.html

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 infraction will be reported through FAIR (Faculty Academic Integrity Report)
- Unless their exist mitigating factors, the most common sanction will be to fail students from the course (category 3 sanctions) for academic integrity violations.

Are these behaviors ok?

- Asking instructors or TAs for assistance
- Using, with citation, code from the provided texts, class website, or lectures
- Discussing, in group projects, specific designs with other group members
- Discussing general strategies with other students
- Copying snippets of code from github or previous semesters
- Using other's code as a hint or template

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

TIPS FOR SUCCESS

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.** You shouldn't shy away from learning difficult things, you should welcome it. Learning doesn't stop when you're done with ECE 391, nor when you're done with school. To stay competitive as an engineer you need to keep learning, but I'd say it's important as a person to always be learning.
- Also, students should not to let the reputation of the class intimidate them. 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)**. Read whatever the class provides at least twice before starting to solve the problem or write code. There were so many times students ask questions that can be answered with a direct quote from the docs we gave out. And also, learn to use search engines too.

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; "It looks like I'm clobbering my stack somewhere in lines 10-20" is more likely to get useful results than "something is wrong with my code".

L2Debug

- Learn to use GDB, don't rely on "printf" to debug, that's what newbies do.
- **Get comfortable with your debugging setup--you'll be using it a lot.**
- 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.

Time management

- **Starting early** is the most obvious but also most helpful piece of advice. If you spend a few hours a day on 391, you'll be a lot more effective than trying to complete a checkpoint within a 48 hour timespan. As a TA, I saw so many groups that were trying to rush to complete checkpoints in a day or two, and all of them looked miserable.
- My advice to ECE 391 students is to **start early** on assignments and stay ahead of deadlines. Bugs will crop up at the worst possible moment
- Everything will **take longer than you think**, so you need to plan ahead if you want to sleep around finals week.
- **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.
- **Plan ahead**. This sounds so simple, but it's actually the hardest thing to do, because of the temptation of cutting corners. It may feel like a waste of time, but it will save you countless hours of debugging in the future.

L2Code

- 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. That will come back to haunt you, especially in Checkpoint 5.
- 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. Instead, trust the primary documentation from the class and Intel. If I were to do anything differently, it would be to plan ahead and to not actually start coding until I really understood what I was doing. Sometimes you gotta just dive in, but there were **far too many times where I started coding and it "worked", but it didn't really because I didn't actually understand what was going on**
- You don't get extra points for being clever: spend more time commenting your code and removing magic numbers, and spend **more time thinking about the architecture** of your OS design.

That Guy

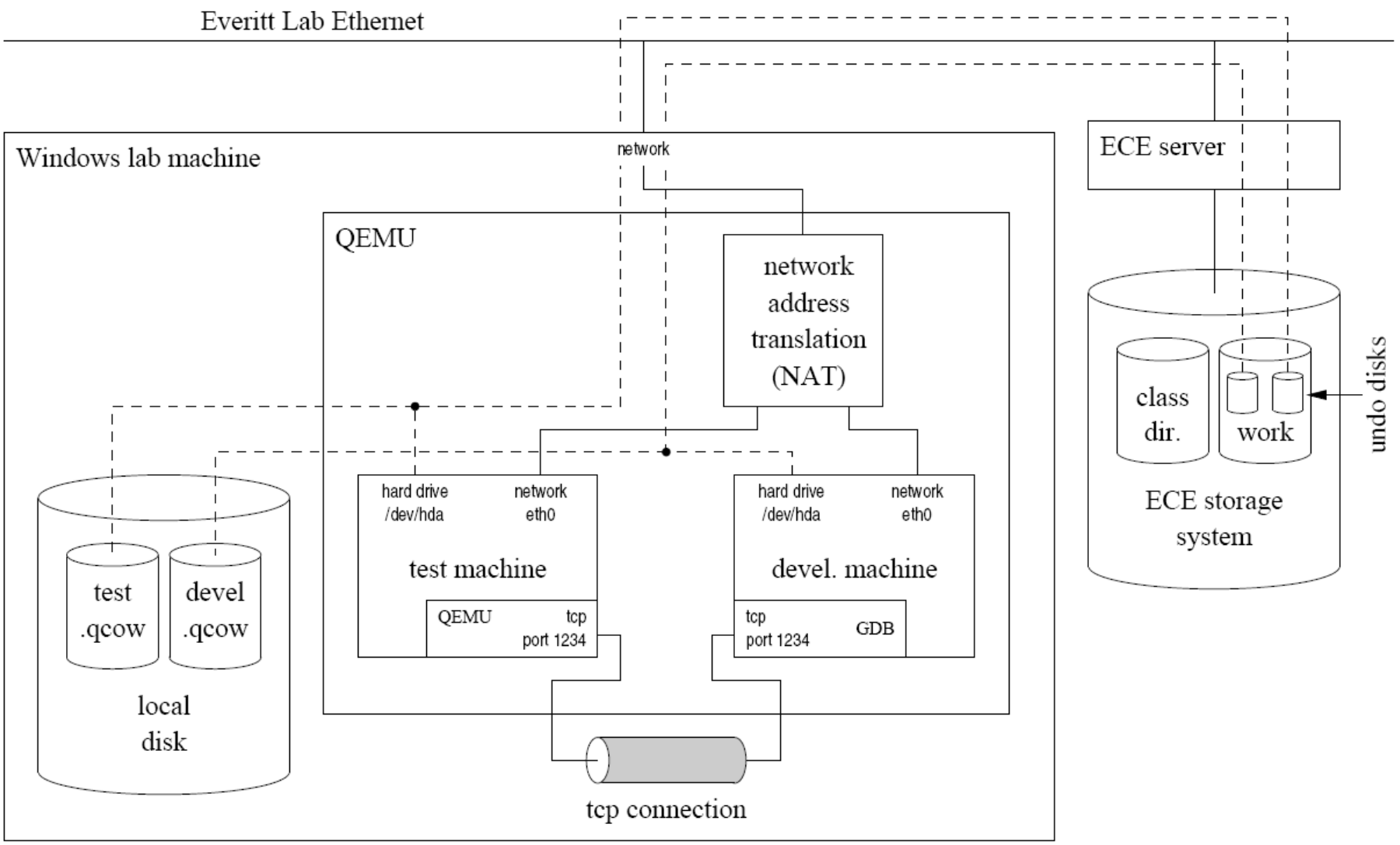
- if you let your teammates do everything, then you learn nothing and the exams are impossible. Don’t be that person. **Help your team and learn the stuff**.
- If I were to retake ECE 391, I would **spend more time getting involved with my teammates** in their every day coding to better understand their day to day work. Casual code reviews and standups, or a variant of them, 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 codebase - even the parts they didn't mainly work on.

Options Available in an Emergency

Emergencies can happen anywhere and at any time, so it's important that we take a minute to prepare for a situation in which our safety could depend on our ability to react quickly. Take a moment to learn the different ways to leave this building. If there's ever a fire alarm or something like that, you'll know how to get out and you'll be able to help others get out. Next, figure out the best place to go in case of severe weather – we'll need to go to a low-level in the middle of the building, away from windows. And finally, if there's ever someone trying to hurt us, our best option is to run out of the building. If we cannot do that safely, we'll want to hide somewhere we can't be seen, and we'll have to lock or barricade the door if possible and be as quiet as we can. We will not leave that safe area until we get an Illini-Alert confirming that it's safe to do so. If we can't run or hide, we'll fight back with whatever we can get our hands on.

If you want to better prepare yourself for any of these situations, visit police.illinois.edu/safe. Remember you can sign up for emergency text messages at emergency.illinois.edu.

ECE391 Execution Environment



Material Review

(from previous classes)

Number Systems

- Base 10 representation (decimal) (0..9):

  $d_n * 10^n + d_{n-1}*10^{n-1} + ... + d_2*10^2 + d_1*10^1 + d_0*10^0$

  Eg: $3045 = 3*10^3 + 4*10^1 + 5*10^0$

  $= 3000 + 40 + 5 = 3045$

-

- Base 2 representation (binary) (0..1):

  $d_n*2^n + d_{n-1}*2^{n-1} + ... + d_2*2^2 + d_1*2^1 + d_0*2^0$

  Eg: $101101 = 1*2^5 + 1*2^3 + 1*2^2 + 1*2^0$

  $= 32 + 8 + 4 + 1= 45$

Number Systems

- Base 16 representation (hex) (0..9,A..F):

  $d_n * 16^n + d_{n-1}*16^{n-1} + ... + d_2*16^2 + d_1*16^1 + d_0*16^0$

  Eg., $3AF = 3*16^2 + 10*16^1 + 15*16^0 = 3*256 + 10*16 + 15 = 943$

Base Conversion

Example: Convert 45 to binary

45 divides by 32 (2^5) once, leaves 13

13 divides by 8 (2^3) once, leaves 5

5 divides by 4 (2^2) once, leaves 1

1 divides by 1 (2^0) once, leaves nothing [done]

Thus: 45 (base 10) == 101101 (base 2)

Signed & Unsigned Numbers

- We need a way to represent negative numbers
- Simple idea: Use the first bit as a sign bit!
  - s = 0: positive (+)
  - s = 1: negative (-)
- *Problem:* There are TWO zeros 1...0 and 0...0
  - Difficult to process negative numbers (special case)
  - There is a better way to handle negative numbers!
- Solution: Use Two's complement
- Numeric Formula:

$$-d_n*2^n + d_{n-1}*2^{n-1} + \ldots + d_2*2^2 + d_1*2^1 + d_0*2^0$$

- Notice the negative sign in front of d_n

Signed & Unsigned Numbers (cont.)

- To subtract A-B, perform A+(-B).
  - the addition operator works for negative numbers
  - first bit still *represents* the sign of the number.
    - s=0: positive (+)
    - s=1: negative (-)

- Three methods to compute a negative number (n) (choose one method)
  - Inverting bits then add 1
  - Take largest number (all ones), subtract n, add 1
  - Scan n from right to left, copy zeros, copy 1st one, invert rest

Review From Previous Classes
Signed & Unsigned Numbers - examples

- Examples (8-bit):
  - -1 = 1111,1111
  - -2 = 1111,1110
  - -128 = 1000,0000
  - +1 = 0000,0001
  - +127 = 0111,1111
  - +128 = Invalid
- Examples: (16-bit)
  - -1 = 1111,1111,1111,1111
  - -2 = 1111,1111,1111,1110
  - -32,768 = 1000,0000,0000,0000

Sign and Zero Extension

- Consider the value 64
  - the 8-bit representation is 40h
  - the 16-bit equivalent is 0040h
- Consider the value -64
  - the 8-bit two's complement is C0h
  - the 16-bit equivalent is FFC0h
- The rule: to sign extend a value from some number of bits to a greater number of bits - copy the sign bit into all the additional bits in the new format
- Remember - you must not sign extend unsigned values

Sign Contraction

- Given an n-bit number, you cannot always convert it into an m-bit number if m < n
- Consider the value -448
  - the 16-bit representation is FE40h
  - the magnitude of this number is too great to fit into 8-bit value
    - you cannot convert it to eight bits
    - this is an example of an overflow condition that occurs upon conversion
- To properly sign contract one value to another the bits that you wish to remove must all contain either zero or FFh

e.g., FF80h *can be signed contracted to* 80h

0100h cannot be sign contracted to 8-bit representation