ILLINOIS TECH

Introduction

CS351: Systems Programming
Day 1: Aug. 23, 2022

Instructor:

Nik Sultana

Slides adapted from Bryant and O'Hallaron

Quick poll

- C experience?
  Which compiler/tools?
- Assembly language experience?
  Which architecture and assembler/tools?
- Experience with other systems languages?

Overview

- Course theme
- Five realities
- How the course fits into the CS/ECE curriculum
- Academic integrity

Course Theme:
Abstraction Is Good But Don't Forget Reality

- **Most CS and CE courses emphasize abstraction**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
- **Useful outcomes from taking CS351**
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance
  - Prepare for later "systems" classes in CS & ECE
    - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Storage Systems, etc.

Abstractions?

- **It means many things!**
- **For an example, let's take "Hello, world!"**
- Three versions of the program: **Python vs C vs Assembly**
- They all give the same output!
- How do they differ in their abstractions?
- How do they differ in the resources required?
  (How do they compare in terms of "efficiency"?)

Do you see the abstractions?

```
%define NEWLINE 10

section .data
  message: db "Hello, world!", NEWLINE
  message_len: equ $-message

section .text
global _start

_start:
  mov rax, 1
  mov rdi, 1
  mov rsi, message
  mov rdx, message_len
  syscall

  mov rax, 60
  mov rdi, 0
  syscall
```

Next time: You'll learn how to understand this.

Do you see the abstractions?

```
%define NEWLINE 10

section .data
  message: db "Hello, world!", NEWLINE
  message_len: equ $-message

section .text
global _start

_start:
  mov rax, 1
  mov rdi, 1
  mov rsi, message
  mov rdx, message_len
  syscall

  mov rax, 60
  mov rdi, 0
  syscall
```

Data types,
segments,
instruction set architecture (ISA),
assembler-specific builtins (e.g., db, $),
syscall behavior.

Next time: You'll learn how to understand this.

Abstractions?

- **It means many things!**
- **For an example, let's take "Hello, world!"**
- Three versions of the program: **Python vs C vs Assembly**
- They all give the same output! **~500** **~30** **~5**
- How do they differ in their abstractions?
- How do they differ in the resources required?

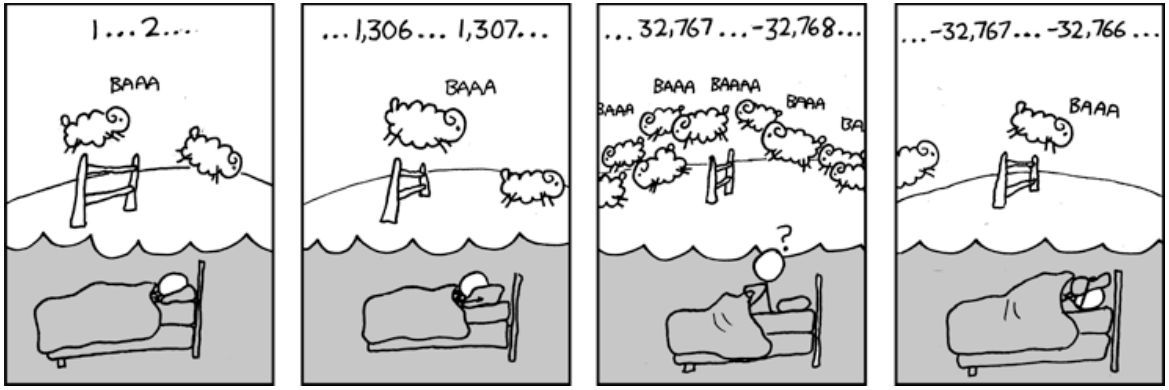
More Syscalls = More Overhead
= Less Performance

Next time: I'll show you how #syscalls comes about

Great Reality #1:

`Int`s are not Integers, `Float`s are not Reals

- **Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!
  - Int's:
    - 40000 * 40000 → 1600000000
    - 50000 * 50000 → ??



Source: xkcd.com/571

- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Computer Arithmetic

- **Does not generate random values**
  - Arithmetic operations have important mathematical properties
- **Cannot assume all "usual" mathematical properties**
  - Due to finiteness of representations
  - Integer operations satisfy "ring" properties
    - Commutativity, associativity, distributivity
  - Floating point operations satisfy "ordering" properties
    - Monotonicity, values of signs
- **Observation**
  - Need to understand which abstractions apply in which contexts
  - Important issues for compiler writers and serious application programmers

Great Reality #2:
You've Got to Know Assembly

- **Chances are, you'll never write programs in assembly**
  - Compilers are much better & more patient than you are
- **But: Understanding assembly is key to machine-level execution model**
  - Behavior of programs in presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage process state
  - Fighting malware
    - x86 assembly is the language of choice!

Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated
- **Memory referencing bugs especially pernicious**
  - Effects are distant in both time and space
- **Memory performance is not uniform**
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.0000061035156
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

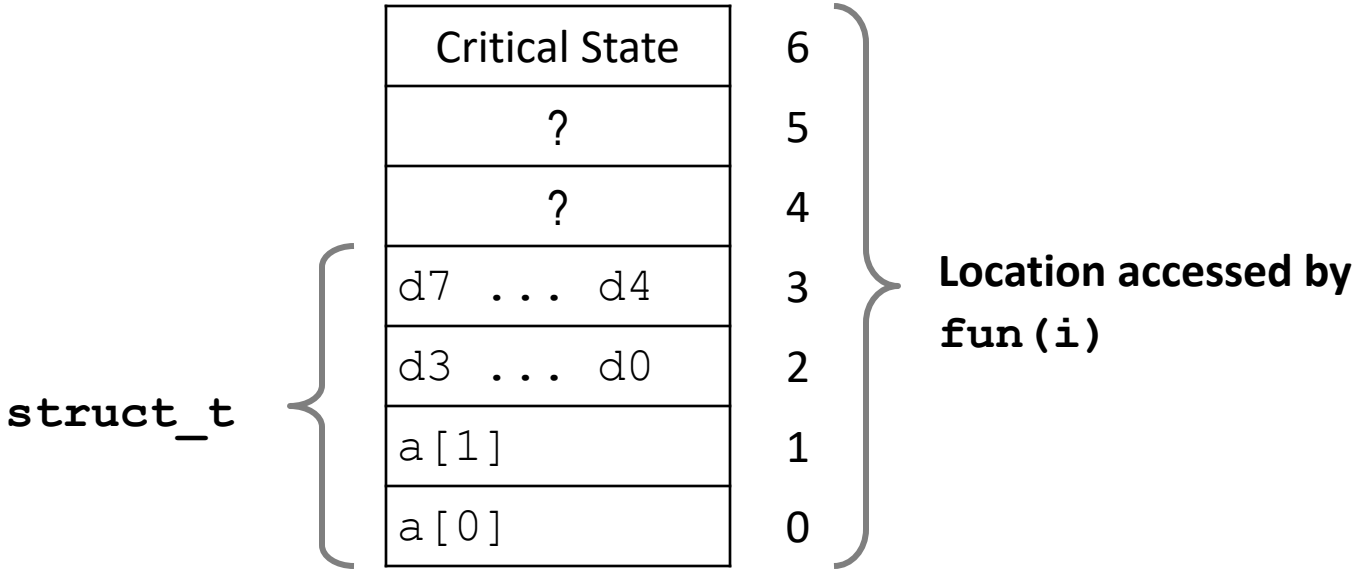
- Result is system specific

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.0000061035156
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

Explanation:



Memory Referencing Errors

- **C and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- **Can lead to nasty bugs**
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated to one being accessed
    - Effect of bug may be first observed long after it is generated
- **How can I deal with this?**
  - Program in Java, Ruby, Python, ML, …
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)

Great Reality #4: There's more to performance than asymptotic complexity

- **Constant factors matter too!**
- **And even exact op count does not predict performance**
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **Must understand system to optimize performance**
  - How programs compiled and executed
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality

Memory System Performance Example

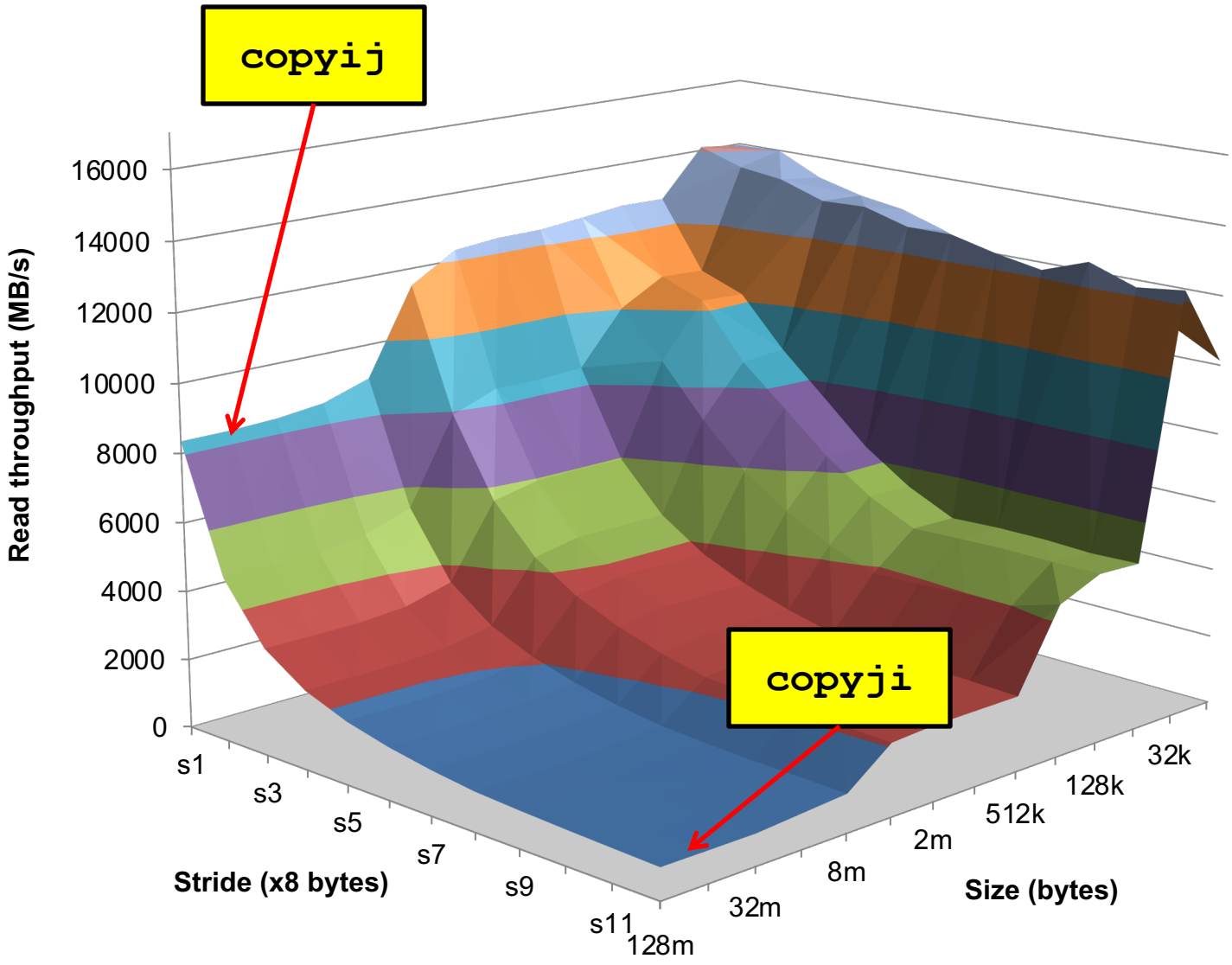
```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms **2.0 GHz Intel Core i7 Haswell** **81.8ms**

- **Hierarchical memory organization**
- **Performance depends on access patterns**
  - Including how step through multi-dimensional array

Why The Performance Differs



Great Reality #5:
Computers do more than execute programs

- **They need to get data in and out**
  - I/O system critical to program reliability and performance

- **They communicate with each other over networks**
  - Many system-level issues arise in presence of network
    - Concurrent operations by autonomous processes
    - Coping with unreliable media
    - Cross platform compatibility
    - Complex performance issues

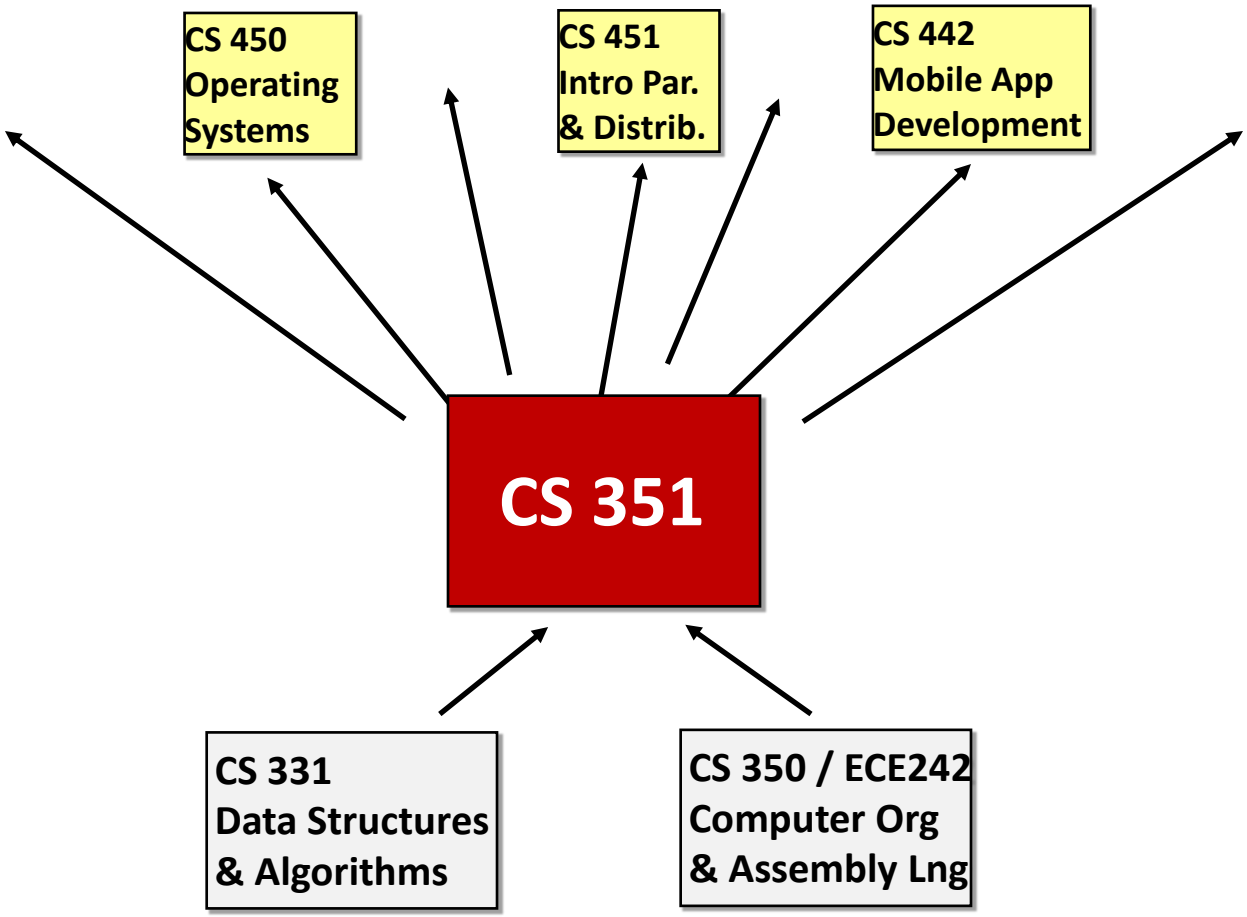
Course Perspective

- **Most Systems Courses are Builder-Centric**
  - Computer Architecture
    - Design pipelined processor in Verilog
  - Operating Systems
    - Implement sample portions of operating system
  - Compilers
    - Write compiler for simple language
  - Networking
    - Implement and simulate network protocols

Course Perspective (Cont.)

- **Our Course is Programmer-Centric**
  - Purpose is to show that by knowing more about the underlying system, one can be more effective as a programmer
  - Enable you to
    - Write programs that are more reliable and efficient
    - Incorporate features that require hooks into OS
      - E.g., concurrency, signal handlers
  - Cover material in this course that you won't see elsewhere
  - Not just a course for dedicated hackers
    - **We bring out the hidden hacker in everyone!**

Role within CS/ECE Curriculum



Why challenge abstractions?

- **Real "full stack" understanding**
  Nuts, bolts, and fundamentals.
- **Tricky debugging**
  (e.g., when using experimental compilers or hardware)
- **Resource-constrained computing**
- **High-performance computing**
- **Low-level programming**
  (e.g., drivers, parts of the OS)
- **Games**
- **Security & cryptography**
- **Research**
  (on all of the above)

Next time: I'll show you actual examples

Example of Why

https://gitlab.com/DeDos/flowdar/-/blob/master/generation/tracer_hooks.c#L99

F flowdar
Project information
Repository
Files
Commits
Branches
Tags
Contributors
Graph
Compare
Issues 0
Merge requests 0
Deployments
Packages & Registries
Monitor
Analytics
Collapse sidebar

```
89  trace_end (void)
90  {
91  }
92
93  inline void
94  __cyg_profile_func_enter (void * callee, void * caller)
95  {
96    struct log_entry entry;
97
98  #ifdef CALLSTACK
99      asm("movq %%rdi, %0" : "=r" (entry.rdi));
100     asm("movq %%rsi, %0" : "=r" (entry.rsi));
101     asm("movq %%rdx, %0" : "=r" (entry.rdx));
102     asm("movq %%rcx, %0" : "=r" (entry.rcx));
103     asm("movq %%r8, %0" : "=r" (entry.r8));
104     asm("movq %%r9, %0" : "=r" (entry.r9));
105     // The argument callee is passed via %rax (instead of %r
106     asm("movq %%rax, %0" : "=r" (entry.callee));
107 #endif // CALLSTACK
108
109   entry.direction = FUNCTION_CALL;
110   // Should not assign caller and callee here, as rdi and rs
    argument passing.
111 #ifndef CALLSTACK
112   entry.caller = caller;
113   entry.callee = callee;
114 #endif // CALLSTACK
115   entry.pid = getpid();
116   // The macro __NR_gettid is found here: /usr/include/asm/u
117   // On Linux x86_64, __NR_gettid = 186
118   entry.tid = (long)syscall(__NR_gettid);
119
120 #if CHECKED
121     int result =
122 #endif // CHECKED
```

Why?

http://www.cs.iit.edu/~nsultana1/student_projects/

Opportunities for Student Research Projects

Info about research projects with me in Fall or Spring semesters

Nik Sultana, Assistant Professor of Computer Science, Illinois Institute of Technology

Last updated: 22nd August 2022.

To participate in the projects described on this page, you must be enrolled as a student in any program at Illinois Institute o

Why work on research as a student?

Because it benefits both the research and the student. The research gains from students' input and energ
covered in coursework or an internship. Depending on how much you engage with the research, it's poss
research artefact, or a code release.

This research experience can be pivotal for deciding what to do after finishing your current degree progra
the technical skills you develop.

If you're thinking of doing a PhD -- or if you want to consider that option -- then doing research as a stude
mindset from coursework. In addition to acquiring this mindset, doing research as a student can help you
to work in, understanding some of the state of the art, and learning relevant techniques. Both undergrad

> If you're considering enrolling for a PhD in Computer Science related to my resea

What could you work on?

When deciding what to work on, you can be guided by your curiosity, what skills you have, and what skills you'd like to develop further.

coming semester I'm offering these student projects **to both undergrad and grad students**. Contact me if you have the necessary skills and wish to find out more:

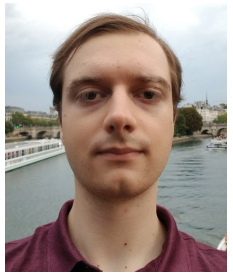
- **Low-level system modelling.** This requires strong C++ skills. Knowledge of computer architecture is a plus.
  If you're an undergrad you can participate on this project by applying to the RES-MATCH program run by the Pritzker Institute.
- ~~**Network modelling 1.** This requires amazing Python skills, and knowledge of networking.~~ (Taken)
- **Network modelling 2.** This requires strong Haskell and/or OCaml skills, and knowledge of networking.
- **Armour R&D.** Engineering students (e.g., ECE) are welcome to contact me to discuss Armour R&D proposals related to hardware and network engineering.
- **Extending the Caper tool.** This requires strong OCaml skills and some knowledge of networking.
  You can try Caper online through the BPF exam site. For more information about how it works, read the paper about Caper.

(The amazing)

Teaching Assistants

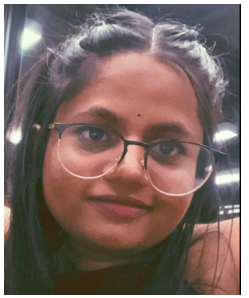
Lab 1: Kirtan Shetty -- kshetty11@hawk

at SB-112J



Lab 2: Alexander Wolosewicz -- awolosewicz@hawk

at SB-112E



Lab 3: Gauri Kumari -- gkumari@hawk

at SB-112F

Lab 4: Caitlin Davitt -- cdavitt@hawk

at SB-108

Cheating: Description

- **Please pay close attention**

- **What is cheating?**
  - Sharing code: by copying, retyping, **looking at**, or supplying a file
  - Describing: verbal description of code from one person to another.
  - Coaching: helping your friend to write a lab, line by line
  - Searching the Web for solutions
  - Copying code from a previous course or online solution
    - You are only allowed to use code we supply

- **What is NOT cheating?**
  - Explaining how to use systems or tools
  - Helping others with high-level design issues

- **See the course syllabus for details.**
  - Ignorance is not an excuse

Cheating: Consequences

- **Penalty for cheating:**
  - Removal from course with failing grade (no exceptions!)
  - Permanent mark on your record

- **Detection of cheating:**
  - We have sophisticated tools for detecting code plagiarism
  - Last Fall, 20 students were caught cheating and failed the course.
  - Some were expelled from the University

- **Don't do it!**
  - Start early
  - Ask the TAs for help when you get stuck

Consider this

- **Employers care about skills more than grades.**
  Cheating can get you the grade, but not the skills.
- **Your job interview has already begun.**
  Take your degree seriously.
- **Getting hired is hard – mis-hires are expensive!**
  Start working on your interview performance *now*.
- **Don't rely on catching up – it becomes harder over time.**
  Making up for lost time is an opportunity cost.
  Instead, apply continuous effort.
- **Best way to learn: engage and exercise.**
  Learning is messy and it takes work.
  But it'll improve your skills. See first point on this slide.

Textbooks

- **Randal E. Bryant and David R. O'Hallaron,**
  - *Computer Systems: A Programmer's Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016
  - http://csapp.cs.cmu.edu
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems

- **Brian Kernighan and Dennis Ritchie,**
  - *The C Programming Language*, Second Edition, Prentice Hall, 1988
  - Still the best book about C, from the originators

Course Components

- **Lectures**
  - Higher level concepts
- **Labs (5)**
  - The heart of the course
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam coverage
  - ~2 weeks each
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- **Exams (midterm + final)**
  - Test your understanding of concepts & mathematical principles

Getting Help

- **Class Web page:**
  http://www.cs.iit.edu/~nsultana1/teaching/F22CS351/
  - Complete schedule of lectures, exams, and assignments
  - Copies of lectures, assignments, exams, solutions

- **Blackboard**
  - Used for announcements, grading.

- **Discord**
  - Used for a/synchronous contact with TAs.

Getting Help

- **Discord: #cs351**
  - Use this for all communication with the teaching staff
  - Caitlin: @Endeavour#8857
  - Kirtan : @kirtan#0856
  - Alexander: @Ruling#1073
  - Gauri: @GauriKumari#1380

- **Office hours:**
  - See course website

Lab Policies

- **Work groups**
  - You must work alone on all lab assignments
- **Handins**
  - Labs due at 11:59pm on the day of its deadline (see course website for timetable)
  - Handin happens automatically on Fourier – just leave your code where it is!

Facilities

- **Labs will use the server called Fourier**
  - `shell> ssh fourier.cs.iit.edu`
  - OTS have provided you with logon credentials – **check that you can access Fourier!**
  - To access Fourier from outside campus, you must use IIT's VPN.

- Fourier is our "single source of truth"
  - If your lab code works on Fourier, then it works!
  - If your lab code works on your laptop but not on Fourier, then it doesn't work!

- **Getting help with Fourier:**
  - Please direct questions to your TA

Timeliness

- **Grace days**
  - **5 grace days** for the semester (across all labs).
  - Covers scheduling crunch, out-of-town trips, illnesses, minor setbacks
  - Save them until late in the term!
- **Lateness penalties**
  - Once grace day(s) used up, get penalized **10% per day**
  - No handins later than **5 days after due date**
- **Catastrophic events**
  - Major illness, death in family, …
  - Formulate a plan (with your academic advisor) to get back on track
- **Advice**
  - Once you start running late, it's really hard to catch up

Policies: Grading

- **Exams (50%): midterm (25%), final (25%)**
- **Labs (50%): weighted according to effort**
- **Final grades based on a straight scale**
  (see course website)

Programs and Data

- **Topics**
  - Bits operations, arithmetic, assembly language programs
  - Representation of C control and data structures
  - Includes aspects of architecture and compilers

- **Assignments**
  - L2 (datalab): Manipulating bits

The Memory Hierarchy

- **Topics**
  - Memory technology, memory hierarchy, caches, disks, locality
  - Includes aspects of architecture and OS

- **Assignments**
  - L3 (cachelab): Building a cache simulator and optimizing for locality.
    - Learn how to exploit locality in your programs.

Exceptional Control Flow

- **Topics**
  - Hardware exceptions, processes, process control, Unix signals, nonlocal jumps
  - Includes aspects of compilers, OS, and architecture

- **Assignments**
  - L4 (tshlab): Writing your own Unix shell.
    - A first introduction to concurrency

Virtual Memory

- **Topics**
  - Virtual memory, address translation, dynamic storage allocation
  - Includes aspects of architecture and OS

- **Assignments**
  - L5 (malloclab): Writing your own malloc package
    - Get a real feel for systems-level programming

Lab Rationale

- **Each lab has a well-defined goal, and is similar to solving a puzzle**
- **Doing the lab should result in new skills and concepts**
- **We try to use competition in a fun and healthy way**
  - Set a reasonable threshold for full credit

Per-lecture feedback

- Better sooner rather than later!
- I can help with issues sooner.
- There is a per-lecture feedback form.
- **The form is anonymous.**
  (It checks that you're at Illinois Tech to filter abuse, but I don't see who submitted any of the forms.)
- https://forms.gle/qoeEbBuTYXo5FiU1A
- I'll remind about this at each lecture.

Stay engaged!

- **The timetable on the course webpage will guide you.**
- **Make an effort to learn C and x86_64.**
  It won't just help you in this course.
- **Carry out the <u>preparation</u> before each lecture.**
  Your future self with thank you!

	Thursday	Friday
	Aug 25 **LEC 2:** C and x86_64 toolchains **Preparation:** Read K&R Chapter 1, and work through Ray Toal's NASM tutorial.	Aug 26
	Sep 01	Sep 02

Welcome and Enjoy!

Questions?