Home
Schedule
Assignments
Recitations (Labs)
Office Hours
Exams
Style Guideline
Academic Integrity
Your Well Being
Ed Discussions
CMS
FAQ

CS4414: Schedule of Lectures (Fall 2021 version; Spring 2023 will be a revised version of this)

All lectures will be delivered in person, and attendence is important -- the slide decks don't really cover all the details we discuss in lecture.

An important note about the books: BO (Bryant and O'Halloren) is used for multiple courses (they explain this in the preface), and hence explores topics not included in CS4414. Moreover, they use C for examples. Despite this, we picked the book because we like their coverage of the topics we talk about, and C is a subset of C++, so their examples work in C++ too. If a section doesn't look like something we talked about in class, just skip that section. If uncertain, just ask us (for example, on Ed Discussions). BS (Bjarne Stroustroup) is a fantastic but very focused text specifically on C++. We don't have a separate Linux book, because all of that material is online. You will also find the C++ reference website (cppreference.com) useful.

Our broad topic is focused on programming "the system", as distinct from writing a program to solve some single problem.

To tackle this topic, we'll need to understand the system that we are controlling: the system we are programming. So in particular, although CS 4414 looks closely at Linux, the bash shell and commands you can launch from it, and at C++ 17, this is not actually a class focused on teaching you Linux, bash or C++ -- we will show you aspects of all three, but we expect you to learn these yourself (with our help and with various pointers we provide).

Lectures 1-7 look at big issues we need to be thinking about: various types of resources (memory, CPU cycles, files...), costs and opportunities they expose, and the many forms of concurrency we see in a modern system, where the operating system might be hard at work, the file system doing various things as well, and where your own code could also be running in multiple processes or using multiple threads. Running through all of this are some unifying concepts that we will revisit throughout the entire semester.

1. Aug 26	Introduction	pptx pdf video	We will explore the different kinds of costs seen with standard languages and systems. BS Chapters 1-3. I'll show some code examples (including one I wrote that

isn't using a very pure C++); you can see the actual code here. We recommend coding the way Sagar does, which is here. A wonderful TA wrote versions in Python and Java for us in 2020 (Lucy Lu). Lucy's code is here and here. Notice how few lines Sagar's C++ code needed, or Lucy's code in Python. Ken's code was faster, but used C instead of pure C++ (this was back before Ken became as much of a C++ wizard as Sagar), dives much closer to the hardware and O/S architecture to take advantage of very low-level features, and needed way more code (plus, he used a style of code that can easily lead to bugs, although in fact his code wasn't buggy). What can we conclude about these language choices?

As a side-remark, after Sagar saw that Ken's "ugly" version was fastest, he modified his more elegant version and gained most of the speed. In fact the version you are seeing is Sagar's version 2.0 -- his version 1.0 was really very slow, about 10x slower than Ken's. *Does this tell us anything about coding for speed?*

There is a famous adage: "Build one to throw away". *What do we learn from version 1, and why do we throw it away instead of just improving it "in place"?*

And just for completeness, there is a third adage to learn too. Butler Lampson, who won a Turing Award for his insights into how to build efficient, clean operating systems software, is famous for complaining about how version 3 (of anything) is always worse than version 2: Full of useless features, often slower, and often full of what could be thought of as the code equivalent of a rhetorical flourish ("fancy language that serves no purpose"). *What would be some examples of things a word counter program could do that would be in Butler's category -- features where Butler would react by saying "Yes, you* **could** *do that. But why in the world would you want to?"*

			the world would you want to?
2. Aug 31	Architecture	pptx pdf video	Modern computers use a NUMA architecture. What does this mean and why does it matter? Skim BO Chapters 1-4, but see note above!
3. Sept 2	Concurency	pptx pdf video	There are many ways to harnass the power of a NUMA machine. In Linux, people used to refer to this as "multitasking", and we'll look at that idea (for example, running a few Linux commands at the same time). But we'll also start to touch on ways that we can use parallelism within our own programs. This will become a big theme for the class that runs through the entire semester.
4. Sept 7	Linux	pptx pdf video	We'll dive a bit deeper to gain an appreciation of what Linux is doing, and how it can actually be "programmed" through scripts that treat Linux commands as programmable components. BO Chapter 7 has some useful context, but in fact much of this lecture is focused on Linux commands, and the bash and Linux help documents and user manual are the best sources of details here. You should definitely read about these, and try them!
5. Sept 9	Memory	pptx pdf video	Data lives in memory, but in a Linux system, even the basic idea of a memory segment turns out to be a very sophisticated concept. We'll have a look. BO Chapter 6, Chapter 9.
6. Sept 14	Systems abstractions	pptx pdf video	Dijkstra was the first to suggest that an operating system "abstracts" the hardware, and that this concept could be carried forward to create a hierarchy. We will discuss concepts of modularity at the level of larger components of a system. BO Chapter 9.
7. Sept 16	Exceptions	pptx pdf video	Everyone is familiar with "normal" control flow. What happens when something abnormal occurs? Interrupts, signals and C++ are all examples of what we call *exceptions*. They illustrate Dijkstra's idea of taking seemingly disparate concepts and unifying them through an abstraction that spans their different functionalities. BO Chapter 8.

A core concept in systems programming is the idea of taking control of all elements of the system. *In a modern computing platform like Linux, all sorts of elements are programmable or controllable, but you as the programmer often need to understand the computing features you are trying to control and you often express that control in somewhat "indirect" ways, for example by coding in a particular style (lecture 8, 9), by "reprogramming" the compiler to generate code for your classes that reflects your own logic and intent (lecture 10), or even by introducing extra layers that redefine seemingly straightforward features (lecture 12).* *In the following lectures we work our way up to this realization, but by lecture 11, on Performance analysis, we see the full picture: if you as a developer can properly visualize your full system, you can use Linux tools to confirm or refute your intuition about where time is being spent, and ultimately use this subtle control features of Linux, the hardware, the compiler, the linker, etc. In fact even the hardware is often programmable!*			
8. Sept 21	Hardware Parallelism	pptx pdf video	In modern systems, the key to speed is parallelism. We obtain concurrency with threads, but the hardware is also capable of true instruction-level parallelism: so-called SIMD computing. How can we help the compiler find opportunities for SIMD parallelism in a standard program? Again, the key turns out to center on viewing the idea of hardware parallelism as an abstraction -- a conceptual building block. BO Chapter 5.
9. Sept 23	Constant Expressions	pptx pdf video	This lecture starts a new unit that dives deeper on sophisticated C++ concepts. We'll begin with a deeper dive into the C++ concept of constants, which have come up in our previous lectures and especially in lecture 9. Constants and constant expressions are evaluated at compile time, and this ide has been taken exceptionally far by modern C++ compilers, to the point that the language has a whole compile-time sublanguage! BS Section 1.6. We will also discuss a peculiar behavior seen with modern compilers when they encounter code that includes some form of undefined behavior, like an explicit floating point divide by zero. C++ and many other languages might *delete* the undefined code block as part of its constant-expression compilation step! This can be a real surprise, and is important to understand. Learn more here

			is important to understand. Learn more here.
10. Sept 28	Templates	pptx pdf video	C++ abstract types are a compile-time concept, implemented by templates. Understanding them is a key step in gaining real proficiency in C++. BS chapter 4, 6-8.
11. Sept 30	Performance: Big Picture	pptx pdf video	We've learned a lot about C++, the file system, NUMA hardware and memories, and hardware parallelism. Performance can feel like a juggling act! In this lecture we'll discuss the big picture for achieving high speed in real programs.
12. Oct 5	Performance	pptx pdf video	Our emphasis has been on achieving the utmost in speed. Can we measure performance? How would a developer zero in on the performance-dominating portion of a program? BO Chapter 5. BS Chapter 5. I should probably comment that even though the slides for lecture 11 seem to focus on gprof, the topic really is a higher-level question. Tools like gprof (and Linux has many of them) are awesome, but they can't even come close to what you need to learn to do at a "visualization" level: trying to visualize how something is executing, and using the insights from doing that to either confirm (or refute) your theories by running one of the tools and seeing if the output matches your expectations. So the theme of stepping back and controlling the entire system (from lecture 10) becomes a theme of visualizing the whole system and using performance tools to focus in on parts that are surprisingly slow -- that deviate from what you expected.
13. Oct 7	Linking	pptx pdf video	When an application is compiled, the compiler transforms the code into an executable. How are libraries handled, and what happens at runtime when a program that uses DLLs is actually launched? This topic straddles the boundaries in an interesting way: It has aspects that relate to the application in C or C++, aspects that relate to Linux, and aspects that introduce a new kind of abstraction, which even creates new "super powers"! BO Chapter 7.

--	**Oct 9-12**		**No class (Fall Break)**
Our next large module looks at threads and thread-level synchronization. More broadly, our focus is on multiprocessing: situations in which more than one element of a system cooperate to carry out some task, using a mixture of methods: multiple threads, multiple processes, perhaps multiple computers, perhaps even attached hardware accelerators that are themselves programmable.			
14. Oct 14	Threads	pptx pdf video	We've seen threads in a shallow way, but this lecture starts a much deeper unit on thread-level concurrency. Creating threads, distinction between lightweight threads and threads with a dedicated CPU. Lambda notation in C++. Taskset command. BO Chapter 12, BS Chapter 15
The in-person prelim will be at 7:30pm on October 14, and covers material up through October 3 -- all lectures up to but not including Linking on October 5 (linking will be included on the final but not on the prelim). Cornell assigned us two rooms: Malott Hall 228 (our normal lecture hall) and Malott Hall 251. Malott 228 will be the main room used -- we plan to use Malott 251 for people with SDS accommodations.			
15. Oct 19	Synchronization	pptx pdf video	Race conditions, critical sections, mutual exclusion, locking. BO Chapter 12, BS Chapter 15
16. Oct 21	Monitors	pptx pdf video	The monitor pattern is a powerful tool for structured control of thread-level concurrency. BO Chapter 12, BS Chapter 15
17. Oct 26	Deadlocks	pptx pdf video	Deadlocks, livelocks, how to prevent them. How systems that are at risk of deadlock deal with this (abort/rollback/retry). Priority inversion. BO Chapter 12, BS Chapter 15
18. Oct 28	Coordination	pptx pdf video	Software design patterns and how this idea can be carried over to coordination patterns in modular applications: Barriers, leader-workers, ordered multicast, all-reduce. BO Chapter 12, BS Chapter 15
19. Nov 2	Multi-Process Systems	pptx pdf video	Many modern systems are big, created by teams and split into multiple processes that sometimes even run on different computers. How do the forms of sharing and synchronization we have learned about apply in such settings? What approaches have emerged as the winners for these big systems?

20. Nov 4	Sockets and TCP	pptx pdf video	In lecture 18, we heard about networking. This lecture will look at how Linux exposes networking via the sockets API. TCP and Google GRPC. How a web browser uses these ideas. These topics aren't covered in the textbook, but you will find a lot of information here and here.
21. Nov 9	Two modern networked file systems	pptx pdf video	In lecture 18, we heard about the idea of accessing a file system remotely over a network. Today, we'll discuss two modern networked file systems. Ceph is an "object oriented" file system. Zookeeper is a file system used as a coordination tool.
22. Nov 11	Prefetching and Caching	pptx pdf video	Why aren't remote file systems slow? This may feel like a dumb question, but when you consider that companies like Facebook are serving content globally with millisecond response times (and what is Facebook beyond a social network graph stored in a fancy file system?) you'll realize this is WAY more tricky than it seemed at first glance. Hint: It isn't just about the network.
23. Nov. 16	Key-Value Stores	pptx pdf video	We talked about networked access to a file system in November. In these two lectures, we'll talk about the MemCached concept and how modern systems manage *really big* data sets.
24. Nov 18	Transactions	pptx pdf video	Classic distributed computing problems. We will talk about the *transactional* model and the 2-phase commit problem that arises. Then we will combine this with a different concept called 2-phase read-write locking, and will see that we can implement the transactional model using these building blocks. This is a powerful and popular way to build strongly consistent distributed systems (not the only way, and if you take CS5412 in the spring, you can learn about others).
25. Nov 23.	FaRM: RDMA-accelerated transactions on a key-value store.	pptx pdf video	This lecture puts the ideas from lectures 22 and 23 together and also offers a glimpse of the cutting edge: we will see how Microsoft used RDMA network accelerators to achieve blazing performance of a transactional key-value service called FaRM. FaRM is used as a component of Microsoft

			called FaRM. FaRM is used as a component of Microsoft Bing, their main cloud search solution. Read more about FaRM here.
-	**Nov 24-28**		**No class (Thanksgiving Break)**
The final module of the course focuses on security. Cornell has entire courses on this topic, so we limit ourselves to aspects of security directly tied to the systems programming concepts we've seen in lectures 1-23.			
26. Nov 30	Security risks in C++ and Linux	pptx pdf video	We will start by discussing classic ways of attacking applications on Linux systems, often via networking APIs that don't adequately check length limits on incoming objects. BO Chapter 3, especially 3.10
27. Dec 2	Linux Protection Mechanisms	pptx pdf video	Having seen a form of hacking attack, we'll continue our pivot and look at other security and protection mechanisms. Access to a resource must be authorized, and the party performing the access must be authenticated. How Linux handles this.
28. Dec 7	Access control abstractions	pptx pdf video	Access control abstractions and the Linux options we use to actually limit access and control the flow of information within a machine, or between machines.

Please note that we have two exams. The first is in person, on Oct 14. The second will be a take-home (online) exam. We will release it on December 9, and the rule will be that you will need to (1) download it within 72 hours, (2) do it without any help except from TAs or Professor Birman, (3) upload your solution no more than 3 hours after you download the exam. We will run an open Zoom session on Thursday from 3-5 during which you can be sure help is available.