October 30, 2023

Week 10, Lecture 1

Plagiarism is plagiarism is plagiarism. I told you what academic dynasty met in the beginning of the semester and some of you are still under the operating. That you won't get thrown out. If you got. This is the copy of my question. Why is popular right? Would it be? And why does it exist right? Do you remember most copy and write exist because we don't actually do anything with the four. We fork and execute run of exact call this result? There's no ever number reason to write memory. That's the answer. That's the answer. That's why it's sufficient. Not just because it's costly. If it's costly, it's still needs to be done. It still needs to be done. In fact, coping right if it needs to be done, it is done. Yes. The reason it exists is because. We don't need it most of the time. Not because it's inefficient, because we don't need it. But I wanna show you some. I want you to look carefully. There are no students names here. Somebody might recognize there, but that's not my point of that. What is being copied? When is it copied? Why is it copied? Right. What is being copied? When is it copied? What is being copied, right? Use this one. There you go. Look at that same almost same thing. I don't know who in this class thinks I am stupid. Or that I won't notice this? Or that I don't care. I clearly made it, made it obvious that I do care. Now here's the thing. Most of the people that answer the questions like this tend to be wrong. They tend to be the worst students anyway. So when they copy and paste and answer, they copy and paste the wrong answer anyway. So grade just goes lower, right? Have this done. K2 Fortunately not in this class. Two people in the second class got zero on there. One of the questions, fortunately it was just one of the questions. So they just got a zero on that one question. Yes. It's a very high pointed question, but. Right another class 6 answers were copied and pasted. Not similar, not kind of similar. Exact, yes. They got zeros on their exams. On their exams, 15% of their grade is a zero. Do not operate under the belief that I do not care or that I will not catch you. You can get thrown up, yes, I'm aware of that. And your visa blah blah blah. If you're an international student, all that stuff is true. I've also warned the shit out of you. Yes, You can't say you weren't warned. Yes. Very likely, although 100 not 100% sure for this glass case. Fortunately, nobody got caught even though there's some obvious places where. People got lower grades basically in this class, right? Not because they cheated, because they copied the wrong thing. They reworded it. Fortunately though, that that was good, right? I'm looking at you and what you call it, because I know I could make him make him give me that bullshit smile. But the point is, it's very obvious, right? It's obvious and it's. It's stupid now. 50% of your grade does not mean you're gonna fail. Right. So fortunately we're not at a point inside your final exam, Final exam before 30% nobody is in this class can pass with 30% of their grade PM0. Alex, right, at 0% people, right? That would require you to get 100 on everything else, and nobody at this point has 100, yes. Alright, so very likely, but not sure we'll have a pen and paper exam. Well, now we're no longer use the Internet for exams. You'll come in, you'll have a paper exam and no Internet. You see? I'll show it to you again if you wanna see it again. Eight people answered that, copying the right thing the same way. We all just found the answer online and then reworded it. That didn't workout for them though, because they of course got the wrong answer. I told you that on the first exam. The first exam we had the same problem. Second exam more the same. Yes, now in the other classes there's out Now you know they didn't even bother to reword it. Right. So they got the zeros. Sorry. You're gonna get a little punished. Punished. It's better than no punishment. Yes. A couple things I want to mention to extra credit. Has not been assigned yet. Has not been assigned. I left the sign in sheets at. Yes, and they should not be here today. I can't tell you why I shouldn't be here today, but. Shouldn't be here today. I'm still feeling it, but. We have 9 recorded classes at this point, guaranteed 9 courses. Next, does anybody remember what's happening Wednesday? Where will I be on Wednesday? Jenna Jodio. You know what that is? That's Texas. I gotta get a passport. Right. Single another country. So I'm from Louisiana, which is next door. So we just make fun like, you know, like you guys might make fun of Iowa. What's Iowa? Stay over there. Or Indiana or Missouri or whatever. Same kind of concept. Actually, I don't know. My cousins made fun of Nebraska because they were from Colorado. They don't know what. Nobody's ever said anything, but Wisconsin is usually a big. Hey, Cortana Country. But anyway, no, I'll be out of town. And then, of course, Thanksgiving. The Monday before Thanksgiving, we actually have class. I'll be in New York. Getting the Turkey ready for smoking. Yes, it's got lung cancer. I'm gonna put it out of the misery. We may not even have Turkey. I was hoping if we have a non traditional open to try lamb. I really like the idea of smoking lamb. You know. But anyway. Give another courses and the reason why I said the count and why isn't counted portant international students have to be in here a majority of in class If it's an in class class the way we can prove it saying class classes by having majority of classes. Thanks. So that's fifteen classes. Right. We can never get to the point where that's gonna be under any final exam actually accounts for. Hopefully we'll get close to the end of the semester and I'll record a week just right. Let's see my face. Look at it. Look at this. Smile when I say that. Yeah, this smile when I say that's not right. And so forth. A couple of things that I want to mention though just briefly. Homework 4. Is do you like Monday or Tuesday? I don't remember today. I didn't get a chance to because I was talking to someone students in class. Unfortunately, I didn't get a chance to go look and see what it would do, but. I should have that in time, but I'm traveling so.

That would not that bad about, but the one where no people didn't even use the two like just ignored the fact that there were two just left. A simple Caesar cipher. I mean some people's homeworks were just like other people's homeworks and it's kind of like being generous here. Five points up here 5 point right. I can't imagine being difficult to find a Caesar cipher all right. A simple one, right? So if you ever feel like I'm giving you something, that's. Easily found online. Read. Read it again. Yes. Also changing every semester just a little bit because every now and again somebody turns this happened on the exam. I don't remember which class but somebody said. There's no setting for RNG only. Right. Remember their exam about, you know, read only. It was gonna be our write only. Somebody put there's no setting for RNG only, but we'll assume you mean WRO only, right? You know why the students said that? Last time I taught this class. I didn't correct that. Oh. But you know what funny part is right? You're still gonna rock. So what the point is gonna make you change your answers. I do change exam questions all the time, but yeah, the other thing I want to mention. This goes from where my international students. So you have me call you a name? Yes, like for instance. Anusha is a perfect example. You know what I call you? Everyone in this class knows, right? Yes, as an example. Everybody knows what I call them. Please do not use the massive long names. Just keep it to like one or two. OK, but make sure like Vignesh is somewhere in here. Vignesh. Just put Vignesh in there. You don't need all the other names. Vignesh and maybe your last name. I'm sorry I don't have my glasses on me like your last name. Like, right, Just do that when end up happening Morality very long and that's fine. No, that's fine. Yours is never confusing. It's never confusing. In fact, nobody in this class is confusing, however, in my later classes. Get a couple of students that are not using names that like one student asked me to call him something and then that's something that's never in the family soon came to say student came to me and said hey I need you to send me why did I get this grade on this homework And I said, well I'll send it to you. And then I I looked at it like. The name didn't match. So I had him e-mail me, emailed him back, and I was like, is this? Is this your? Is this you? Yes. His last name is somebody else's first name. So it wasn't obvious and so. Yeah, please do that in the future.

Couple things I want to mention too about bird concept. Oh. Mention this already next Wednesday. We do not have flash classes already. Record classes not yet recorded, but it will be recorded before you sit down for class on Wednesday. Yes, I'm not sure how far I was gonna get. Instructions ready, Ready is actually pretty simple threading concepts, including terminology, benefits and cost. Most of you would have experienced writing concepts in your operating system class, so the core idea is pretty much there. This is a new material, but how do we actually do it is in this language We'll talk. Well, let's talk about training models. Training models themselves are actually super simple. Don't complicate it. Operating system does things or doesn't. Either the application has to do it or the operating system. Right, it's not massively complicated. One of the things we're gonna counter pretty quickly is you're going to want to simplify your life as much as possible. And when I mean simplify your life, I mean simplify your life. We're gonna get really complicated really quickly. OK, So what do we do? What do we do when things get really complicated? Abstract. Abstract. Right. Take out the details, make sure you understand this part, how it relates to that part. Yes. Well, let's get her see. And parallelism. We'll talk about race conditions using synchronization, using mutual exclusion. We will not do like philosopher problem. What is it, the dining philosophers that's thinking is right? Two people have two chopsticks between them and pause for a second to think and then they pick up a chopstick. Is alternate almost immediate deadlock? Yes. Look it up. It's a classic synchronization problem. It really is like you. You're gonna do some synchronization. But anyway, race conditions are actually pretty simple and we'll discuss deadlock. It's avoidance. We're really gonna focus on deadlock avoidance, right? This is not a distributed systems class, so we're not really just worried about. Issues with that, however, we're gonna generally try to avoid it. We'll talk about the ways in which we avoid it. And then finally we talked about the threads API, including thread implementations and using the API now. It is dreadfully simple. There isn't. Much to do. Create the threads. Actually pretty simple. Terminating, canceling, pretty simple couple. I mean maybe six method names you need to learn maybe. Right. And then, you know, dealing with race conditions, deadlocks, locks. Right, There's nothing. Code wise it's complicated here. Except keeping. Each other from screwing things up is really hard. And you go. What do you mean? So now we're gonna deal for the first time. We're gonna deal with shared memory, right? Theoretically, when you dealt with sockets, you have not sockets. Excuse me. Forks, right? If you have a forking problem, you wanna go see the doctor, right? There's a bad joke there. But anyway, with forks you do have shared memory, but only for a brief amount of time. The second any changes made, there's never. Cross cross contamination. Nothing that Nathan's. Four version can do to my fourth version to cause my 4th version to be incorrectly updating something, right? If he changes something, he gets his own copy of everything. Yes, I still have my copy. Yes, they're they're never any changes with that. I'm sorry. Sort of like where where essentially prevents one person from screwing up another person, but the difference with GitHub is when there is an interference. Right. What does it do? GitHub actually manages that. So it lets somebody go in and say which change is better, which change, which, you know it's over with. And it's sort of sort of fixes the zipper right? Where you know the zippers overlapping, right? Fixes the zipper right. So puts them back in line. Then you get a finalized copy here. We're gonna be dealing with shared memory now. Your memory is gonna allow me to write your variables and you can write my variables and then we could just have really confused things. The examples I'll use are very contrived. Yes, the examples I use are very very contrived. I'm gonna use a counting variable. Bank account balance or something along those lines. And I'm going to. Updated several times and have multiple threads update the same variable multiple times and you see pretty quickly that without some sort of synchronization they overwrite each other and you can get incorrect errors or you get errors, but the bad news about these errors and this is where it gets a little rough. So with forking. You can do some checking and so forth, and then you can report the errors and so forth, but with threads. They're overwriting each other, so. It becomes very difficult to see who made whose fault it is. Because they're all just writing each other. Becomes more and more complicated. So that's what I was talking about, keeping things simple, right? If you keep things simple and abstract and keep them abstract, it makes all of it right. And what I mean by that is make sure that when you are writing to variables, you know why you're writing to them and. If it's gonna affect other threats as well. Alright, so we'll come back to this slide again buyers compiled program process, it's an instance of a running program that unit of activity within a process. This has its own virtualized processor stack, processor state and everything. Smallest unit of execution execution that is scheduled by an operating systems process. Yeah. Each process contains one more thread. If you only have one thread, there's a single unit of execution and only one thing going on at once. This, we think are the single threaded processes. This is what you're used to, right? Book starts referring to in chapter Whatever chapter is Chapter 7 and a single threaded processing. Every program be written up to this point is a single threaded process. You still think of it that way, right? But if you have more than one thread, then there's more than one thing going on, Then we refer to that as a multithreaded process. Now hopefully you've seen some of this before. Virtually every viewer application you'll work with should be a multi threaded application and you go. Why is that unless it's a really really simple model view, controller type situation? If you have something running in the background it will cause the gooey to freeze. So you want to? Unfreeze it and that's great. As an example. Anyway, OS extractions, modern operating systems provide two fundamental virtualized abstractions. Yeah, virtual memory associated with each process, and you have virtualized process associated. Together they give the illusion that each process is the only running process. OK. Process. You get the virtualized memory, yes. Multithreading benefits, so we get the programming abstraction. So multiple threads doing multiple work. It seems like a natural way to do things. Imagine batch processing and I got six images to work on, right? I gotta do some heavy duty processing with this image, right? Maybe I'm doing a curl process? Right. Blurred it or so. Some of you aren't familiar with image processing. Maybe I'm blurring it or doing some color correction. Process like go each pixel and I look at all the pixels surrounding it and make some decision about which each pixel value should be based on the surrounding thinks. It's a very computation heavy. Thing to do right? I can go do one at a time, right? It takes a long time, right? Could take hours, right? So I could actually break it up and run each one have its own thread, right? Or more realistically, I could actually break it up. Because of the way it works, I can actually break up each image into multiple threads. Each pixel makes a decision about what value should be based on its surroundings. But this guy's pixel value, this guys 3 spells can be computed the same even though they're next to each other, they can be computed at the same time. Right. So it it's provides programming abstraction, which is a natural problem to some solutions. You're gonna you can guess with your homework for what you're gonna least do a little bit with homework 5. You'll feel it coming, right? I have to go through a multithreading directories with threads. Yep. Yep right. Parallelism is improving throughput. This throughput mean? Let's suppose. We need to do laundry. Does everybody do laundry? OK, I know these two guys. They're supposed to smile at me. Everybody does laundry. They need it, right? Does anybody actually like doing laundry? Jesus Christ, of course it's you. Don't I hate larger. I really do hate. It's in the basement and, well, I don't like. The basement 's not that big deal with allergies. I hate going down to the basement, but anyway. Do you put the clothes in the washing machine and you put the clothes in the dryer? Right. You can do that. What if I put the clothes in the washing machine, Then when I put the clothes in the dryer, I put new clothes in the washing machine. Now the washer and dryer going at the same time. Instead of going cereal do the entire job, I'm gonna break it up into two components and they have those two components working at the same time and that will increase throughput more done quickly. More done, faster. Is that better? Do you have laundry in the same room as you See, that's why. It's just my only quiet time where the kids leave me alone, I get to watch TV and I'm folding. It's my alone time, so I love it. Everything. I know you have kids, right? Like when that load larger comes out, that was magically that entire load is all kids clothing. Tiny human sucks much worse. But for me, it's not my flight time. I'm actually doing laundry. Watch The SpongeBob SquarePants or some other bullshit. But anyway, it improves through book. You can get more done faster, yes now. Sometimes. This is the only way we can get things done in a reasonable amount of time. If you think about image processing, the kernel processing I was talking about. We're actually looking at 10 pixels for each, not ten pixels. Even if I'm being generous, right? Free across. 6. 7-8, so eight pixels. That's all the directly surrounding with. We go even further out, that's even more those need to be looked through. Add it divided and that's your answer. For each one you have a modern computer of 2000 by you know, 10,000 pictures, right? 10,000 pixels you have to go through takes a long time. So what happens without parallelism? It gets the point where it's like, what is it even happening? You can't run a job and then just expect this thing to be finished tomorrow, right? What if you screwed up? You know what happens. You screwed up. Oh, that worked for the last two days. We did a job. The Army Corps of Engineers. We're doing flood just I'm not shitting you. I'm really not shipping. We were doing flood level elevation for certain types of flooding events prior to Katrina. In the New Orleans area, right? We did not model the things that actually happened, but that's what we're trying to do. Certain times of rain events, certain types of things, whatever. Anyway. As a result, all the modeling it is a simulation that produces a map. Result that involves a flood elevation for every location. That involves a lot of simulation. So this runs on a big giant Bank of computers, yes? Even with the big Dale Wolf cluster. This thing would take two or three days to run, yeah. One of my buddies, not me, thank God. Because he's running it now on our computers. He's running it on theirs, The Army Corps of Engineers, right? He screwed up. Screw up. You just move the same job he just finished. A couple days. And they were like, why did we do this? And so everybody went to figure out who did it. I'm like. It's not true. Right. And they did, look, they did just do that. But the point in that made is, is that it takes getting used to run this simulation, right? Because the number of distinct points in which this simulation was important was thousands, hundreds of thousands, right? Collision for city right? Even block by block. This is a massive number, but anyway. And there's lots of variables that go into that. Battle Fluster 64 ish computers. Running for three days and it's massively parallel. That's not possible, right? That would take months if it wasn't in a cluster, yes. So parallelism is very very. Improving responsive responsiveness. This is the most common. This is probably some of your first experiences with threading if you've worked with GUI applications in the past. If you work with GUI applications in the past, anything that involves anything that involves, like any type of processing, the GUI freezes up instantly. So what do we do? We have a gooey thread and then we have a process thread. Yes, you have to kick off something, You do it. Another one. We're seeing more common these days with back end web servers, full stack web development as you receive back end servers. Have a what we call a process threat, a mechanism to kick off an extra thread so that it can return the response immediately and separate the response from the work, Yes. We're not seeing that. That's been existing for a long time. That's what we're hearing now, blocking IO. If you have blocking IO, you have to wait for things like threads, sockets and all those things. Contact switching. Thread thread switching is much cheaper process cross stitching and so we see it more often for that reason. The other thing is memory savings. We use shared memory because we can utilize shared memory. We can save a lot of overhead from having a copy of it. Hi. It's considerably more difficult. Then doing it without. Everything up to this point. Nothing can override any method, nothing gets so far. Can override any variables and so forth so. Awesome. Great. The general problem is when you have concurrency with shared memory, right? If you don't have shared. If you keep everything separate. So that my thread can never interrupt with the other thread, right? This idea of mutual exclusion. Works out great. Sometimes you can do that. Your homework, you'll be doing that. Shake your head. You will be doing. You'll do fine. I'll be fine. It's always fine. I just make you suffer a little bit. But how do we solve it? So not just synchronization word Sometimes algorithms, if you think about a way we can order these things, we can actually do it without. Just the way we just organize the algorithm that could actually benefit from threads, right? So we think about our kernel processing. Mechanism for. I'm trying to think of a name of it. Blurry and type filter. Gaussian filter is similar but you have to actually but the point is gonna make is. Colonel seen in image processing type applications takes a long time. All map applications work the same way, right? You have massive number of things you have to have in order to get. One side of the map to the other, right? But. Can I organize the algorithm in such a way that when I make a decision about a pixel? I can make that decision. Separately and distinctly from the Pixel next door. Yes. Right. If you think about the way blurring actually works, the most part, yes. Is it gonna be perfect? No. Close to work, yes. Sharpening and some other tools do not work as well. There are other techniques that you can employ to separate them as well, for example when you're using sharpening tools. It's harder to separate a pixel that is side by side. But the top and bottom pixel is fine. As an example. So you can separate the algorithm and think or think about the ways in which you can break the algorithm up to solve the problem more efficiently. Suppose you're looking for 10 names in a list. You have 10 names in the list, and you're looking for these people in these giant text files. And then you have to store some sort of frequency associated with each one. You can look at each text file and then modify the list of files, right? You certainly could, right? At work, you're guaranteed serialization, though Yes, you can synchronize, because this is what size can synchronize. It, of course, and then only the people, right? But what if you hit it multiple times? Then lots of things are waiting on you to get it back in that thing. What if I told you you pull out each name? Thread for each day. Now the friends would never modify each other ever, at any point, yes. You can do without synchronization completely, right? Just if you think about the way the algorithm works and how you can actually provide medical exclusion without synchronization, then do so. You will improve performance. Bad news is the threading model and synchronization strategies have to be part of the design. Adding synchronization things because something's broken is usually break something else. I'm serious, OK? Think about the problem. Think about the synchronization that needs to go along with the problem. Synchronization by itself is actually pretty simple. There's not a lot of code associated with. You have a lock and you unlock more locks with the code. Right? It's. Except, however, what actually makes it difficult, if anything makes it difficult, is the fact that it's usually those locks are not thought out very well, yes, and we're going to get into more problems than that. Alright, 3 miles purchased. Implement threads to pencil and functionalities provided. Usual or kernel space. So one to one threading or kernel level threading. There's a one to one relationship between the kernel provides and when it's consumed. That means there is a kernel space thing for each thread that exists. Now into one threading or user level threading. User space is where threading is implement. There's little or no support in the kernel that requires significant user space code to make it happen. Benefit is that context switching is generally cheaper because we don't require the kernel to do anything, we can just do it all. Into M fretting or hybrid kernel and user threads are used and mapped into. So we have a number of girlfriends and we have other girlfriends. We have user friends. Now I think I highlighted it because the Unix we don't care too much, but this is important from a. What's what? What's the where I'm looking for? From a high level perspective, different systems will produce different things. Unix is actually usually 1 to 1, so it's usually the top, but you're not guaranteed that however. Different operating systems would be different things. Writing pattern the first step in development of trading strategies to consider threading patterns. Spring pattern Do you wanna use? So there are a lot of abstractions, implementation too. They're really only two core patterns we need to think about. We have threads per connection. For each thing that has to happen, we create a thread for it to happen. Unit working society to thread that previous responsible for only that unit work. During these execution connection because server you connect to the server and the server has a thread dedicated to you, that's why it's referred to as a thread per connection. It's usually necessary because we have reading and writing happening. From that socket, and we might have reading, writing from other sockets, so we need to actually separate that so that makes sense. Next. When we connect to a server. We'll talk about this later and when we get to servers, but when we connect to server we have two way communication, yes? When you when you talk to your server, right? You work in the food industry, right? When somebody talks to their server or used to, I don't know if you still do, somebody talks to your server, they tell you what they want, the server tells you how much it's gonna cost, yeah. No. Yeah. Don't you know, that's so much fun? I used to say that in the story, people used to look at me like I had six heads. I didn't do it to older people because they would get mad, but younger people would be like yourself. I didn't mean to tell him where I was. That's good walking, moving, but they would be like what? Anyway, no. You talk what happens. You talk, Server talks. Now we think of why do you use serving inquired all the time. We overuse this analogy and the reason why we overuse it is because it's built to work that way. You want it to work? What has to happen? We talked to the server. Server response. We talked to the server. Server responds. What happens if we start yelling at the server? Karen. What happened? Server might start yelling at us. It's communication happening the way it's useful, no. But you can't actually have two threads, one thread doing the four communication, one thread to the back. Table. That absolutely is a beneficial way to do things. Not usually OK very few performance gains can actually be made from that, yes. Since so much about threading is generally just waiting on IO we try we're gonna decouple that waiting from the friends, right? When we think about the guys and the things, most of that is something happens we have to communicate to a server. I created a thread. The first time I created a thread was to deal with the server communication. Was communicating with the server, but this is the thing. Do the same thing it was happening. Did you even freeze? And that's because processing was happening the background, including blocking IO, and that froze the screen every time. So we can decouple that completely from anything involved with IO from the main thread of the execution process. Issue all IO asynchronously and use Multiplex IO to magical control. We are gonna use Multiplex, is that OK about that? After callback the Multiplex. Back-to-back. It will do that, but essentially we're gonna handle all the AIO IO. Why? Yes. The other thing is it gives us a chance to work with friends without just working with a new IO. What gives us a chance to work with friends from a low level without us focusing on? Multiplexing and so yes, currency parallelism, concurrency similar to two more friends to execute an overlapping time periods. This can occur without parallelism. In fact the first time that this people how to use virtual machine with one processor. And so as a result of that, because of the way the system worked, threads would always operate in this fashion. They would go, somebody would go, then the next guy would go, right? Schedule meeting out. Just get accur without parallelism. Concurrency itself is a programming pattern. In fact, there are books on concurrency patterns that you can refer to. We only talk about two because we don't care. I mean, this isn't. Concurrency. We're just gonna talk about threats. But their books on the programming patterns associated with the version is it's a way to approach problems. So whenever you get to a problem, whenever you get to a way to approach problem computer science, you find books of people, people that wrote books explaining how you can solve this, this, this and this and this problem doing it. And that's what it is. But it does not actually mean that they're going to be two things happening at once. It could be we're gonna do this for awhile and we're gonna do this other one for a while and we're gonna do this other one for awhile, right? Could either leave without? Overlap, yes. Now parallelism is ability to execute two more threads simultaneously. Parallelism is a specific form of currency requiring hardware that allows for two threads to operate at the same time if you don't have multiple processors. Yes. Can interleave. But they can never actually happen at the same time. OK, hardware features achievable through concurrency. Quick point. Does concurrency require parallelism? Parallelism require concurrency? Yes, for hopefully obvious reasons, yes, if you don't have the ability to have two things executed. Overlapping time periods. Can you ever have those two things happen at the same time? No, right? But that is the key thing. One of the things we always think about when we think about concurrency is we think about we building build and concurrency so we can make sure that two things happen at once. But that's not usually what guarantees, which usually guaranteed as they can. Now. How many computers do you have? Modern computers, right? How many computers do you have that have more less than one processor? Virtually none of them have anymore, right? They can't actually speed compute processors up anymore faster than they could build more cores on the same processor. So the now the way to do things is concurrency but. Yes, you should all add up hardware to do it, but that wasn't always a given. Certainly work on servers with processor best race conditions. It's a concurrency that introduces most of the pain having to do with, right, right. You're talking about threading your eyebrows earlier. I do not know if it involves pain, but I would assume it does, yes? Does it? It does hurt. Another reason not to have it done right. But since threads can overlap in their execution, it becomes not deterministic. Threads share resources, so active memory accessing memory gets to be like a race where what happened depends on who gets there first. Yes, this is what we're referring to from a race condition standpoint. Whoever gets there first gets updated. What is somebody slow updated? But the guy who got there second, if he's a fast updater, he's gonna update it and that other one might be lost. It gets to be a problem. We're gonna see examples of this hopefully next class, which I will record. But. Need to figure out what. I'm very busy. Yay. Can't wait. What you doing tomorrow? I thought I knew. Was that the day off in the morning? All right, See if you're afraid of the cold. You live in Chicago. I've seen the way you put on cool, Dude. This guy doesn't own, like, a big giant Coke. He's like, he's from Chicago. He's this tough. The other day when it was like 50 outside you shivering this year? Literally, like we talked about this in the dead of winter. He has the same basic jacket he has all year. But anyway. For those that don't know, his birthday is the day after Halloween, so I don't think he likes to do anything for Halloween, which is the point. But for those of us that have kids, we have to trick or treat. Yes, we go cows, the house and they ask for candy, trick or treat, smell like feet, give me something good to eat, right? Something like that. And then they eat candy and so forth. Dress up. But the point is, is that between today and tomorrow I have to report this. Figured out I just stay up late. I'll do it. But race conditions are hard to detect. This is a bigger problem. Now the problems I'm gonna show you are simple. They're designed to show a race problem. Race condition, right? They're designed to show and exemplify it. I'm gonna create a bank account. Type class. And I'm gonna have a random generator that randomly decides whether we're gonna deposit or subtract money. A certain number of times and you can compute how much it should have at the end, right? You're gonna be wrong. Right. And that's the problem. Formally, erase condition is a situation which unsanitized access to the shared resource by two more threads. Risk issue bites you have question. Race condition by itself does not yield an actual problem. Who gives a damn if you the first that first modification made? You have to have modification involved here, which leads to a situation where water more threads within a correct data. Erroneous. You know what that means? Most your programs are erroneous. How do you not know what that means, Alex? Stop. It just means it's wrong. It's producing the wrong result, yes. It's always, not not always obvious, so I always want to working with a. Very friendly application, so many of you know. We did a minesweeper applications with Stanley Ocean 4 pictures. Pictures. Because of the way threading model works, and because of the safety aspect of the thing they actually were recording, every time they found something that needed to be reviewed, they were fined, usually two or three somethings. And the reason for that is. The race condition. They're all fine in it and to be safe. All the other things they were all marking them, but the problem is what? It's the same thing. So all we did was put a guard in there that if it's within what, 4 feet, 5 feet? Don't mark the second one within 5 feet of another one. Why is that? What are we doing it for? No, What are we doing, the mines from 4:00 to 5:00? And you're not gonna get too much. A visual, right? You have to send somebody down there to actually look. If he's been this far away from it. He's gonna look around, and if it is too, he's gonna see him pretty immediately. Yes. And so that is what we decided that was OK. That was our delta. To still be safe, we don't wanna leave mines hanging around, just hanging around somewhere. Because you'll hear about the rats. In New York, they're pretty bad too, man, talking about that. I mean, they're like cows. They do. They're like little. I thought, like cat ran across the back of my head one time, but it was, it was definitely right. Because it was big, big sucker. I know, I'm aware. I was going down to do laundry and I heard something jump over here from one to the wall to the other to the wall. Scream like a 5 year old. Come back inside and I was like, did you guys hear me scream? And my wife was like, no, did you scream? Why did you scream? So it says all right, but anyway, no, no. They train rats in Indonesia, Indonesian countries, Polynesia as well. Old minds exist all over the place. They train rats to go find them. Bad news is the rats. We can get happens that when it finally yes. The good news lots of this we longer live in a world of warfare where mines are produced. Not working anymore. Now the US can do some things that are pretty cool, but we're we're like the only ones. Yes. And you're like, what do you mean we can make detonating mines that we can detonate whatever we want to? Yeah. So they're not, they don't get left behind for kids to find these Polynesian countries, right? We can do all sorts of things, but if a country like Russia decides to do these and not do the not in pulling the thing, then you end up with really nasty things happening, so. Trying to make plastic ones that you dropped from the airline airplane. Field. Mass produced mines that are not to kill a person, but we don't really want them to die. By the way, you don't want the soldier to die. You know, like. I'll leave you with this lasting thought. It's almost Halloween. The soldier is more expensive to take care of. They just his friends aren't gonna just leave him on the field and as a result of that. Hey, get more slim all down, right? But it also. That's good. Let's hope that supply line they have is good enough. Not as terrible as it is, but anyway, that's what I was referring to. Birthday to the most common form and we prefer that data race data rations are used to manage. And the most straightforward to manage. OK, there are very few race conditions that aren't involving that aren't based on data manipulation as a result of the data manipulation. Yes. That's easy, right? You can't get out processes and so forth. That's. That might be another race condition as an example, but the most common one that produces erroneous program behavior is. So the window in which a race can occur is referred to as a critical region. So the critical is gonna happen here. You know modification is gonna happen here. Now this is the all the code is visible. You were the developer. See the code. You know the code, You know where The thing is gonna be modified, You know what's gonna be modified, and you know why. This is where the area. Pretty simple, right? Nope, not actionable. In fact, you add in a whole bunch of additional problems. But here's some critical region examples X + + + + X = X + 4. So before you answer answer topic, none of these things are atomic. Jesus Christ. None of them are top. So before you start saying things are atomic, be aware that 99% of programming languages do not have statements interatomic on their own. You know, wait, what? X plus plus is a shorthand way of writing X = s, yes? Add one and stores it. OK, it's not because some are using. This is gonna be wrong. Yes. But anyway, these are critical reasons. Samples. You could block access to these. When we see my examples, I'll do something literally similar to this, a little bit more complicated. I'll try to put into more realistic example, but this alone, if I had 50 threads doing this, you're gonna end up with a wrong number. Alright, the fundamental source of races is critical regions, which are window during which you correctly. Select. Correct program here requires This is the one where we're gonna go. That way, if I make you go single file, this data race no longer occurs. And what happens? Correct behavior always, yes. So we must synchronize to ensure that the right has access to the critical region. We refer to these operations as atomic if they are indivisible. We're gonna make it atomic. No, it's atomic by default, OK? What we're gonna do is sensually is we're gonna make an operation, whatever that operation is. Could be plus plus X. We're gonna synchronize it so we can make a group of statements if we want to be. And that means that either happen. Or they don't? Right. But more importantly, we don't care about whether they happen or not. It's not a transaction, it is in fact a. Atomic means we're just going to make it single file through this part. Let me go the other way, alright? We're going the wrong way. I'm going the wrong way. I used the critical range. Oh, critical. I missed. I skipped this one. This is what you're gonna see on the example. Double withdraw. Withdraw some money. And you can see I gotta go get the balance. Remove the amount out of the balance. Return the amount yes. At some point you're gonna end up with a number. Why can't number like this, right? There's no way? I'm checking the balance to see if it's great in the amount. At the same time, remove the amount from another balance in a different thread. We will remove. We'll get through that if statement on one. And then we'll remove the amount from one and then remove the amount again and thus end up with a negative balance. So you're like, I can't happen, Yeah, again, we're gonna talk about how to prevent it. Alright. Some mutexes. The common mechanism for synchronization is a lock. You've been in Paris. You put locks on a fence. Right. They don't recommend you doing that because they. Bridge bridge overweight doesn't break the bridge, but it breaks. Usually makes the. What you call it sag The things they're attachment to. Which is very bad, doesn't it? Second and 4th, the bridge isn't gonna fail because the locks on it. Hey, somebody telling me that? Anyway, it's just city Bridge, right? It's not like, you know, anyway. A lot, right? We're gonna we're gonna lock access and what are we gonna lock? Lock it. And then we'll go to the end and then we'll unlock it at some point. Now allow somebody else to get answer that critical region. Basic idea, however nothing's magical outlooks. You can unlock it anytime you want. You have the key. The threat behind me has the key. It's the same person, it's just you're the developer, right? You're creating 50 threads. At least 50 threads all have the same keys, locks and all those things, right? So nothing magical, nothing physically enforces it. You don't just magically get oh, I'm declaring this to the critical region and I'm blocking now. It's not how it works. It does work that way in Java and some high level languages right away. You can literally do that. There's a synchronized keyword in Java and C# I believe as well. I don't know if it's synchronized. I think synchronized. I think in C# you have to use an object, but whatever locks are basically what we refer to as a gentlemen's agreement, we're identifying an area in which this is critical. So we're not gonna unlock it by default, so I lock it. Before I check the balance. Then I checked the balance greater the amount, then I take balance equals balance minus amount and then I could unlock it again. And by just unlocking it. Somebody else can go in and check. It is no. It is eliminating the need, the things happening at the same time, yeah. There may be lots of good reasons to actually. So with lock system, we're only locking A brief amount of time, and so as a result we can actually still increase throughput, just not as much as maybe we did before. Now if you block everything all the time, then you're just detect. What should you ever do? And that's what we're getting at here. As you move further in, don't forget that race conditions are based on data and it's manipulation many threats. Although we lack critical regions of code, we need to ensure that we focus on locking data. Data race conditions are the reason why we have. Quarantine. Nothing good can happen like, right? Stopping that if you lock everything so nothing bad can go wrong. Extra work for new work concurrency. So we add threads, but then we get race conditions. We don't want race conditions, so we add new taxes and then we get deadlocks. You can run into a situation in which a lock. And then what happens? Thread grabs a lot, very brief, grabs another lock. And then they're both trying to get those Alright, I got one worse for you. I'm gonna have to pick on somebody else. Let's go with poor Nagin and. So you're a big natural. Trying to remember your name. So the hashing stands the door now this time. They're not like you guys. Nothing like you guys. They're the exact opposite. Y'all are super nice. So Cortana goes, no Mahesh, why don't you go and Mahesh goes No pornography, why don't you? Guess what happens? The same thing. Neither one of you go. You're too nice, yes. What should happen? In either case, one of you says no, you should go, the other one goes, and then they go, right? That's what they mean by a gentleman reading, right? This is what we mean by a gentleman's agree. If you're too nice, you can run into the same problem if you're doing some situation where you're checking to see if a problem exists before you do it. If two things are checking, they can both find that same condition and us. Get stopped even though neither one is actually injured. To the thing, OK? So we need to focus on deadlocked word avoidance. Avoiding deadlocks is important and the only consistent, safe means to do so. Sign locking, care for. Do careful. Let's suck. Don't you? Doesn't it? Don't you wish I had some special golden king? I had some golden king with a round back end. It was perfect shape. And then you had like 3 little things and you said, here you go and I hand it to you and this is how you just avoid, right? But it's not that doesn't work that. Every problem is different, and every problem may have its own problems associated with deadlock and deadlock avoidance. However, most of them can be just reasoned through. Yes, one particular kind of deadlock is what we refer to as the deadly and Braves. OK, this involves 2 lots. We'll see this one an example. This was usually pretty easy to implement. OK, but one thread acquires 1 mutex, 1 lock. And then tries to acquire mutex B. Another thread acquires mutex B and then tries to get mutex A. Guess what happens? There's a timing scenario. You could draw timing diagrams is different class but distributed systems in which one grabs mutex and the other grabs used XP and they need each other. They can't go. They're stuck. Forever. If you just simply reorder them so they all go get A and then try to go get B, that never occurs. Little bit. Algorithms can be managed this way very easily, depending on what the problem is. But you really try to avoid friends deadlocks, excuse me? Whenever possible, right? Whenever you think about mutexes at any point, if you think about mutual exclusion, or you think about locks or you think about anything, what should you be automatically putting in your head? Avoid deadlocks, right? Otherwise it gets really ugly really quickly. It provides airline primitives that enable threatening but the bulk 30 libraries in districts. What? Pausing standard created threading library and developers call this the positive threads or either it's almost everybody first. Thread API is very large, big contains everything needs to build multi thread. Everything is in three dot H and every function begins with P thread\_All functions can be broken into two main categories. Thread simple management. Great thread Destroy Fred, cancel thread, those things. And then synchronization. If you're gonna use P threads in the sense it's a separate library, it's automated with a P THREAD flag. With GCC, compilation step will change as a result of what we're doing as a result. Yes. Hey create threads we have to conclude pH. We have Pete thread it takes the thread. THP list and it takes a void start routine which. Returns boy and takes Boyd and arguments. Wait, what? It looks ugly, I know. What? Provided by start routine. Start routine is your function that's going to be performed when you start the thread, yes. Restore the thread ID and the pthread T. That's what the thread T is for, the thread ID which you can use to cancel it or wait for it. Change default behavior of the new newly newly created threat. So if you wanna change the behavior, we normally just leave it null. I've never even seen anyone modified, so we usually that ones not start with routine looks like the following. Start. Look like this. Does that's the way it works. Is function pointer is gonna allow us to? Use this P thread create to automatically create a new thread, yes. Right ideas. You can go get it. It's a A thread analog PID P thread itself P thread equal. You can compare to. It's not a number, so don't try to just print it, it's not usually that useful. Do do I do wanna? I don't know. Have a simple example unfortunately. So we're not gonna finish. I wanted to finish and I figured I would not finish in the next class. See. Hold on. Can you close? Yeah, we should be able to get close alright. Cancelable. Whether or not previous cancel depends on a state or not. Cancel will state as you can say cancel state. If you're in a cancel state, it actually puts cancel request on hold, right? So nobody can cancel you cancel type either cigarettes or deferred. Sorry. Any threads. Threads can terminate on some instances. If thread returns from start routine, it's done. If a thread invokes the thread exit, it's done and it gets canceled. It's canceled. All threads terminate the following the process returns to the main function eventually. All threads related, all threads will end, the process terminates to be exit, all threads end. It's like they're they just go away. Not like they finish. OK, be clear about this. Not like it finishes, it goes away. Yes, the process executes a new buyer, right? In any one of those situations, all threads just die. Alright, terminating threads. If you wanna terminate yourself, just P thread exit. You can give it a return value P3 and cancel. We don't cancel, I don't never cancel, but if you ever wanna cancel you can't. We talked about cancel state either asynchronous or default and asynchronous killed at any point after cancellation request. It deferred threat can only be killed the cancellation points which represent safe points which are critical reasons defer. However, again, we don't cancel very often. Will allows you to do. Type allows you to do is when you cancel a thread, you need to cancel a thread for whatever reason. It makes sure that whatever you're writing or whatever you're doing will actually make sure that that's not going to leave you in a nasty state. Joining threads. Joining is the waiting for a thread. You'd be able to join. You give it a thread and then give it a return value. Upon call, we wait for the indicated thread. Once it terminates, the waiting thread is woken up and returned back. Is not null. Provide the return. We say the threads are joint, we don't wait on the thread join with the threat, OK? But it is a blocking call it will actually block alright so i have a question if you guys give me but we may have them detached which remembers them no longer join. Detached. Now what does detaching do? Detaching allows it to continue running. Yes so if you don't accept to join it you should be catching you just give it a thread and you touched me taxes for all their power they're relatively computers