ProgrammingParadigms-Lecture18

Guest Instructor:All right, everyone, Jerry's out of town today so I'll be giving the lecture. And we have two handouts today, two very small handouts actually, the section handout and the solution for tomorrow.

So I believe last Friday Jerry started the ice cream store simulation problem. This is a really huge problem. It's probably bigger than any single synchronization threading problem that you saw so far in the lectures, and it's the merger of two or three final exam problems, so it's pretty involved. So try to go through it as clear as possible but feel to ask me any questions if you want.

All right. So let me just start from where we left off last week. So we were talking about the ice cream store and we have four places basically in the ice cream store. We have the cashier which is they are basically to handle the billing issues and we have the customers. We will say we have ten customers and each customer is gonna ask for a random number of cones – of ice cream cones from one to four.

So the least we could have it ten cones and the maximum is 40 cones. And every single customer because he doesn't want to wait for every single cone to be done sequentially, then he's gonna fight of a clerk or dispatch a clerk for every single ice cream cone that he wants. And so we will have clerks that are making the ice cream cones. We have 10 to 40 of those. But the thing is every single clerk because really giving the ice cream to the customer it has to ask for the manager's inspection to get an approval for the cone that he made.

So the clerks have to get the manager's approval, and the manger that is only a single manager, only a single cashier, and if the manager disapproves of the cone then the clerk basically has to redo the cone and ask again for the manager inspection. The manager, he doesn't like to have two clerks in his office at one time. So basically we can't have more than one clerk contacting the manager at some point.

At the same time – so this actually starts – you can start to see the communication that is going between those guys. We definitely can see that we have clerk threads, right? Because every customer – every single customer would potentially ask for one, two, or three, four clerks. At the same time, we have ten customers that are running in parallel. It's not only a single customer at a time. So we have threads here, we have a thread for the cashier, a thread for the manager, a total of probably 52 possible threads, okay.

You can see the communication going between the cashier and the customer. You don't expect any communication between the cashier and the clerks or the manager, right? You can see communication between the customers and the clerks depending on how many clerks he fires off, and you can see definitely a communication between the clerk and the manager, right.

So you want to start thinking about what are the constructs that you will need in order to maintain synchronizations issues between all of those players in the game, okay. So let's start thinking, just try to brainstorm about it and then we can look at the code. The thing is the customer comes in and he wants one to four ice cream cones and he will fire clerks. After he gets all his cones he will go to the cashier, right? But he cannot go to the cashier until all the clerks give him back his ice cream cones.

So you can start to see, like, this kind of 1 to N communication that we talked about, right. You can see that this guy is gonna request N clerks and he has to wait for the N clerks to get the job done before he can go to the cashier, right.

You can also see the communication between the clerk and the manager. So no two clerks can go to the manager at a time which means that there should be this kind of lock around the manager's office, right. And so only one clerk can access the manager's office. At the same time the manager will be waiting all the time because he has to wait for all the ice cream clerks to be done for all the customers before he can go home.

So the manager will be in kind of a loop just waiting to be requested for an inspection. And once he finishes the inspections he gets back the result to the clerk and he keeps waiting again until all the potentially 40 cones are done and inspected. Make sense?

So the clerk will basically have to request from the manager who's waiting. So there's the manager waiting for the clerk to request. There is also the clerk waiting for the manager to finish his inspection, right.

So we have this kind of binary rendezvous again thing going on in here. You see that? Now there is another constraint on the customers going to the cashier at the end. So the customers are waiting for their cones. And after they get their cones they just go to the cashier in a line and this line has to be maintained on a FIFO order, so a first in, first out, right. And we have to also think about that. How are we gonna insure that the customer that arrives first at the cashier gets really served first and leaves first, right?

You will see, again, the kind of binary rendezvous thing between each customer and the cashier because the cashier has to be waiting all the time for a customer to request the service or basically a customer to show up in line. And after a customer shows up in line he will be waiting for a cashier to finish handling his billing and let him go basically, right. So any question on how the setup is for the problem?

All right. So let's talk about the main function. Let's see what happens in the main. Okay, let me write them in here. All right. So we have the main function. I'm not gonna write this argument but it basically has to maintain the total number of cones, right? The main function is basically responsible for spawning all the threads that we need.

We'll see that the customers are responsible for spawning the clerk's threads but other than that everything gets spawned in the main function. So the main function is gonna basically – you have the total number of cones which is required to spawn the manager because the manager needs to know how many cones there is because he can't leave until he's done with all the cones, right.

So we do the normal stuff, we entered the thread package and we sent up some semaphore that I want to introduce right away. But just bear with me until I get to this point. And then we basically spawned all the customers. So for int I is equal to 0, I is less than 10, int I plus plus. What we want to do is we want to get a random number of cones for every single customer and then spawn the customer and add this number of cones to the total cones that are in here.

So I need to initialize this [inaudible]. So we have an int num cones is equal to a function random integer. That gives me back a number of cones between one and four and this function is in the more concurrency handout slide. I'm not gonna get it.

Now once we have the number of cones for a customer then we can just start spawning the customer because all the customer needs to know is how many cones he's gonna order, right. Make sense? So we got a thread new, debug name, the customer function, and this customer's gonna take one argument which is the number of cones, right. After doing that we have to add this number of cones to the total of cones because we ill need to pass this total number to the manager eventually, right.

So total cones plus equal num cone. Now this is gonna be done for all ten customers and this way we spawn all the ten customers that we have in our simulation. Now we need to model the cashier so we have a thread new, the cashier, and we don't give him any parameters. And we also spawn the manager so we need thread new, the manager, and we pass to the manager the total number of cones, which is single parameter, right.

Now we're done with that, we just run all the threads. After we run all the threads since we did set up the semaphores in here and semaphores, again, remember this is a construct that is appointed to something that is allowed in the heap. So you have to always remember to free your semaphores, okay.

So you go and free – that's called a function free semaphores and then you go and end all of the semaphores that we're gonna introduce, okay. And then you return zero. So this is basically your main function and I don't think, like, in here we're just spawning threads. You don't see any kind of communication going on. We didn't do anything yet, right? We're just starting.

So I was kind of thinking about where to start. Typically the sequence, the logical sequence is the customer because that's the customer that goes to the store basically to buy ice cream. But I think it might be really confusing to start with the customer. So I will go with the manager because it's the easier one and then maybe we see the communication between the manager and the clerk, right.

And then we try to link it to the customer and the cashier because the customer will also have to interact with the clerk. So let's go to the manager. The manager and the clerk. I just described a lot of things going on, right. A lot of basically synchronization things that we have to take care of. We need semaphores in here, right. There is like a [inaudible] I've just explained. We need to know after the manager inspects a cone whether it succeeded or not, right.

So that should be something communicated. It is some sort of communication. So because there is a lot of variables, there are so many variables and there are many threads or accessing all of them, we again do – we access them globally. So we'll have a global variables, global semaphores. There'll be a lot of them so in order to make it understandable we're just gonna group them in structures.

So we'll group all the variables that have to do something with a manager and a clerk in one structure and call it inspection and we'll later on do the same thing for the customers and the cashier. We have another structure with all the variables for the synchronization and we'll call it the line, all right.

So let's have our first global – so we have the struct inspection and, let's see, the manager's gonna inspect everything that the clerk shows to him, right? So we have to need basically a Boolean to show us whether the cone passed or not, right. And I'm gonna be very informal in here. I'm gonna give you the initializations but all the initializations are basically done in setup semaphores, okay.

So we're not gonna go through this function. But this pass is gonna be false. Now what else do we need? The manager has to keep waiting until it's requested, right, for inspection? And also the clerk has to wait until it knows that the manager finished its inspection, right?

So we need some kind of semaphore for rendezvous the semaphore is gonna be – let's call it requested. And so this is gonna be signaled by the clerk to the manager that it requests an inspection, right. Make sense?

All right. So this semaphore is gonna be initialized to zero, okay. So basically the manager what it's gonna do is basically gonna wait on the semaphore which means it's gonna block, okay. Until a clerk comes and sends the semaphore so it basically wakes up the manager. Make sense?

All right. So we have the semaphore requested. We have the semaphore finished and we also understand that once the manager – once the clerk requests inspection it will wait on this finished semaphore which means it's gonna block until the manager finishes the inspection and then signal the semaphore and then it wakes up, okay. So what else do we need in here? What else do you think we're missing? Okay, let's start working and see what we miss, okay. Let's start working on the function and see what we need.

So we'll have the function manager. So that's void manager, the manager takes an N which is the total number of cones. So let's call it total cones needed. And here is what the manager does. I'll have two variables, one of them is required, the other one is not.

The first one is total, let's call it num approved. So this is num approved which is initialized to zero and another one which is num inspected also initialized to zero.

So what we need to do is the following. While the number approved is less than the total number needed then you basically can go home, you have to keep waiting and keep inspecting, right? So while this is the case what you want to do is the following. You want to wait until a clerk requests your service, right? Okay, so you want to wait on inspection, let me finish this here. So you want to wait on inspection, not requested, right? Once you get a request – yes, go ahead.

Student: Which are not needed as [inaudible]? [Inaudible] not needed?

Guest Instructor:Those are not needed?

Student: Yes.

Guest Instructor:So did you see when that spawned the manager thread? We passed the total cones and the total cones were the total number that a customer's wanted. So that's the total number that basically the manager needs to inspect. That's total number of cones, yeah, that all the customers will buy, you see.

Student: Oh, okay. So that's the total number of cones.

Guest Instructor:Yeah, I'm just calling – I mean, I'm just changing the name of the parameter but that's fine, okay. Any other questions? All right. So you want to basically wait on the request for the inspection, okay. And then once you get a request for an inspection you basically want to inspect the cone, right? Okay.

So what you do is you have your – once this is done you basically are inspecting a cone so num inspected will go up and you have to check if you're gonna approve this cone or not. So we simulate. This is modeling, right? So we have a random function that says sometimes, yeah, we do approve, sometimes not, okay.

So we will say inspection not passed which is the Boolean in here is equal to some random chance. It returns one of the two things, zero or one. And again, I'm not writing this function but you understand what it's doing, okay. So again remember I'm assigning this to dot passed because this is the Boolean that I'm using to communicate between the manager and the clerk, okay.

Now if you really passed – if inspect got passed then what you want to do is basically increase the number of approved. Make sense? So num approved plus plus. Now we finished your inspection, okay. The only thing he has to do but what you know is that the clerk is potentially waiting on your finished. So you want to signal finished. Any questions? Yeah, go ahead.

Student: Yeah. Are we using global variables mean or –

Guest Instructor:Yes, we are.

Student: Okay.

Guest Instructor:So this structure is basically a new one.

Student: Okay.

Guest Instructor:Okay? And everything inside is basically initializing this function [inaudible] semaphore, okay. And I'm giving you the initialization. So all these are gonna be semaphore new, okay. All right. So after we do that what's left off is just a signal, the inspection not finished because you know that the clerk is gonna be waiting on the semaphore, okay. And that's your manager and that's the easies function of all.

Now let's go to the clerk. Now any question on that before I move on? All right, so the clerk. The clerk – we'll pick some parameter. I'll worry about it later. Because this parameter's gonna be rated to the customer and I didn't write the customer function yet.

So, let's see, what the clerk needs to do. It needs to make one cone and have the manager inspect it, right. It needs to repeat this cone until it passes, okay. And once it passes it just hands the cone to the customer. All right.

So, let's see, the clerk basically starts with a cone that doesn't pass. So Boolean passed is equal to false. And why you didn't pass you basically need to make a cone, right. And then request the manager to inspect it, okay. So let's see, we can potentially have up to 40 clerks requesting the manager to inspect their cones, right? Okay? Does this give you any hint about what we're missing here?

Student:A binary lock.

Guest Instructor:Exactly. So we need the binary lock. We basically need to lock the manager's office, right, so that we can insure at any point in time only one clerk can get in this office, okay. So here we need another semaphore which I will call lock, and this semaphore is gonna be initialized to one, right. Why is it initialized to one? Why not zero?

Student: So it can get in the office.

Guest Instructor:Exactly, so that at least one clerk can get in. If you initialize it to zero then nobody's ever gonna get in the office, right. So you want one clerk at least to get in. And you want only one clerk to get in. You cannot have it zero. Zero is a deadlock, right, because nobody's gonna ever be able to get in. You cannot have it two because two means two clerks can potentially be in at the same time, okay.

All right. So the first thing that you want to do basically is to acquire this lock, right? So you want a semaphore wait on inspection dot lock, and again, there are two scenarios.

First of all is that lock is still one because nobody did the wait before in which case you'll be able to proceed. The other scenario is that this is zero in which case you just block on it and wait until it is signaled, okay.

All right, so you wait on this lock. After you wait on this lock you basically need to request the service of your manager, right. Okay? So what you want to do is semaphore signal, inspection requested. Am I going too fast? Is it clear? Okay, so we signaled the request and after the – yeah?

Student: Some [inaudible] to one?

Guest Instructor:Excuse me?

Student: Would it say requested to one?

Guest Instructor:It would, yeah. So request is originally zero, right? The manager, which I just wrote in here, it doesn't wait on requested which means it blocks. So the way our semaphores in the library that you are provided the semaphore doesn't go to a negative value, okay.

So it always maintains a zero or a positive value. So if it's zero and you try to decrement it, it doesn't not allow you and it blocks. So you keep waiting there blocked until somebody makes it positive. So you can decrement it to zero, okay. And this is how the blocking works, right.

So what happens here is that the manager would be blocked if he starts before the clerk because what will happen is the manager will try to decrement the zero semaphore but it will block and then it will wait until some clerk comes and does signal the inspection request which will make it positive so the other one will be able to proceed and the command does. Do you see how this works? Okay?

All right. So you did signal request. Now after signal request the manager will get the request, right, and will start working on this. What you need to do at this point is wait until he finishes. Make sense? So you just –

Student: It's really [inaudible] is like if you have inspection request of him and if you lock up a manager why is it he requested?

Guest Instructor:That's a good question. Let me finish it and then I'll answer this question, okay. I'll just go on because, like, I want you to see the whole picture after we finish it. So we will basically wait until the inspection is done. So after the inspection is finished and once inspection is finished you basically want to know the result of the inspection, right? You want to know if your cone passed or failed. Now how do you know that? Which field tells you?

Student: Inspection dot passed.

Guest Instructor:Inspection dot passed, right? Now you want to read into passed whatever the manager passed in inspection dot passed, okay. So you want to set passed equals to inspection dot passed. Now you know that if passed now is two then you're not gonna do this anymore and you're gonna move on, okay. You're done with this point but remember that you were locking the manager's office, right. You no longer need to lock the manager's office. Make sense?

So you want to signal – so you waited on the lock in here. You want to say that it semaphore signal inspection dot lock. There is something missing here. I will finish it when we talk about the customer.

Now back to your question. So the question is the following, why do we need to have lock and requested. Why do we need those two? If lock already grants us access to the manager, right? So think about what would happen if you don't have them, okay.

So let's say you don't have the lock. It's obvious that you need it, right? Because you will have two clerks to go to the manager at the same time, right? Now, think about if you don't have requested. Then what will the manager do? Not at all, like, basically the manager will not wait for any inspection, right? It will go on and proceed with passed, give it a random value and go on and keep going in the loop, right?

Now which clerk will be checking its own inspection dot pass you don't know, right? This lock is crucial and it's crucial that you maintain this – this is maintaining this kind of critical section in here, right?

This lock is extremely crucial because basically you don't want to release this lock until you read the value of inspection dot pass. Note that this value is shared among all the clerks and the manager, right? It's a single value. And because we allow only a single clerk to be with the manager at the time then we can safely assume that the value in here will be up by this clerk that is with the manager.

But you have to insure that no one else can be in that portion, okay. You have to insure. So this lock insures basically that you have this critical section in here. It insures that you can read this passed before you release the lock and before any other clerk can be with the manager.

Because potentially let's think about it differently. What happens if you switch those two lines? What happens if we first signal the lock, basically release the lock and then check the value of what's in inspection? What's in passed? I'm sorry.

Student:[Inaudible] inspection dot passed.

Guest Instructor:Potentially, you're not sure, potentially. What happens is if we had those two lines kind of flipped what would happen is probably you could have the inspection dot lock signaled, which means that the lock of the officer of the manager is no longer locked and any other clerk could get in.

Now if you're Fred, which is this clerk, Fred, gets pulled out of the processor at this point it is very possible that some other clerk could get into the managers office, show him his cone, get his passed value of right original pass value before this guy gets back the processor. Do you understand that? All right?

Now another thing, let's see, what would happen if the manager was the one to signal an inspection lock? Like he would say, "Oh, here, let's give the control to the manager." And the manager has his office lock and he could just unlock it. What is the problem with basically moving this line from here to the manager? You see any problem with that?

Student: Isn't it the same person.

Guest Instructor:It's exactly the same thing. It's basically the manager is gonna unlock the inspection lock. So he's gonna unlock his office and we don't know if this guy actually did execute this line or not, right? So, again, the lock would be opened, the door would be opened, any other clerk could get in, could override the passed value before this guy gets to here. Make sense? Yes?

Student: What is – so the manager waits on inspection dot requested. What if the manager just waited on the door being unlocked? Because it seems like once you unlock the door you make your request at the same time. It's like the same thing.

Guest Instructor:So what you want to do is the following, you want to have the manager waits on the lock.

Student: Yeah.

Guest Instructor:And you also have the clerk wait on the lock.

Student: The –

Guest Instructor:You see the problem with that?

Student:– clerk waits on finished. The clerk doesn't even –

Guest Instructor:What would happen is you have a manager waiting on lock. You have two clerks signaling lock and getting in. You see the problem? The thing is binary rendezvous it's always you need those two things because one is waiting for the other to finish something and the other is waiting also. It's exactly the same idea of the Buffered Writer and Reader which is also generally called the Consumer [inaudible] Problem.

So you have some consumer that consumes some [inaudible] of something, another consumer that consumes the same thing. Now this is exactly the Buffered Reader and Writer. And what happens is somebody's waiting for the other to produce and the other is waiting for him to consume, okay, because there is some shared source in the middle. So you always need those two things to maintain basically the two – you see, the two basic synchronization of the two agents. Whoever is producing waits for the other to consume and the other way around. Make sense? Any other questions?

All right, so we're basically done with the manager and part of the clerk. We'll have to revisit the clerk when we talk about the customer. But before we move to the customer then now we're gonna be talking about the relation between those two guys. And we'll also talk about this guy, the relation between those two.

Let's first start with the easy part. Let's see where it's put here. So we have a void customer and the customer takes the random number of cones that he needs to order, right? So he takes an int num cones and let's see what he does.

The customer gets in the store he is browsing. This is just wasting processing time. Nothing, right? He's gonna spawn N num cones clerks to do every single cone for him, right? So he will basically need to get num cones clerks, like if this is three clerks to work on his cones and wait until they're done. Okay, before he can move on to the cashier. Make sense?

So this means that we will need to have some semaphore and why we need to have it because we know that we need to wait for all the clerks, okay? So we need the semaphore. Let's call it clerks done. And this one we initialized to zero again, okay.

And then for int I is equal to 0, I is less than num cones, int I plus plus, what you want to do is basically fire each clerk for your cone. So you do a thread new right at the end, and then you fire the clerk. That's a debug name. The function is the clerk, right? And now we need to pass to the clerk the semaphore because you will wait on the semaphore for all the clerks and you want every clerk when he finishes the cone to signal this back to you which is equivalent to handing – to giving you basically the cone. Make sense?

So you want to thread new a clerk passed to it one parameter which is basically the clerk's done semaphore. Now this takes us back to here, okay. So this clerk takes a semaphore, let's call it sema to signal. And once it's done making the cone, inspecting it, approving it and it's all done then you basically want to semaphore signal the sema to signal. Make sense?

Now that's for the clerk. Now we're done with the clerk. This guy just spawned num cones clerks, right? He needs to wait for all of them. So he saw last lecture how we do that. You just have another for loop for int I 0, I less than num cones, int I plus plus. You want a semaphore wait on clerks done.

Now we know that this guy will never get past this point until all the clerks are done with all his cones, right? At what point you just could basically go and free this semaphore because you won't use it anymore, okay.

So you could just go here and is it free semaphore or semaphore free? Semaphore free. Okay. You can double check that. Semaphore free, clerk's done. And then you walk to the cashier. Starting at this point you will see how the synchronization works between a customer and the cashier and I think this is the most tricky and the hardest part of the whole thing. Yes?

Student:[Inaudible] the semaphore wait on the for loop [inaudible]?

Guest Instructor:That's a very good question. So the question is the following, why don't we include the semaphore wait call and the for loop in here to just have a single for loop that has basically the thread new and the semaphore wait. Anybody has an idea? Yeah, go ahead.

Student: Because we'll – the clerks will only be spotting once at a time?

Guest Instructor:Yeah, any other ideas? Anybody [inaudible] that? So what happens if you do that is the following, you get a clerk and you wait on done, right? Now you did actually initialize clerk's done to zero, right, in semaphore new. Once you wait on this you basically block, okay.

So you will be blocked in this line. You want be able to go any further in the for loop. You understand what I'm saying? Until this guy who you just spawned does a semaphore signal on done so you can get to the next one. So basically you'd be doing it one by one, okay? Yes?

Student: Can you make this [inaudible]?

Guest Instructor:Right. So the question is the following, can we make the semaphore and a shared integer value variable or a [inaudible] to variable. And so this takes us back to the problem of [inaudible] end use, okay.

So the main reason why we using semaphores or locks or those basically concurrency constructs is that we need to insure atomicity. And by atomicity, we mean that we want to decrement the value or increment it and check its value in one operation or in several operations that are guaranteed to be done atomically. So if you have an integer, the thing is you will increment or decrement it and then you will check its value and between the two you could get of the processor, okay. Yes?

Student: Yeah, how does it know what threads it owns? Like, if – say you've done the for loop and you went out of bounds by one, so you did like num cones plus 1. How does it know what threads to wait on like –

Guest Instructor:How does it know what threads?

Student: Well, like you say semaphore wait, clerk's done.

Guest Instructor:Yes.

Student: And you’re waiting on the semaphore but, I mean, you’ve created that semaphore four times already for the four different threads.

Guest Instructor:That’s correct. You don’t really care to know which thread is gonna signal it. You don’t care to know or at least in the – in terms of the function itself you don’t care to know which one is gonna signal you as long as you know that somebody’s gonna signal you.

Student: So probably one of the threads you’re waiting for on semaphore wait could be some other customer’s thread that was a clerk’s done?

Guest Instructor:One of the ones in – no. Know that I am actually having this as a local semaphore in here.

Student: Okay.

Guest Instructor:So if any other customer has also another local clerk’s done that’s a different issue. It’s a different semaphore basically. All right? Any other questions?

All right, so you recently walked to cashier and from that point on we’ll be dealing with the cashier and the customers. Now, I was just saying that this is a tricky spot and this is the tricky spot because of the constraint that we impose which is to handle all of them in a FIFO order. Basically whoever goes in line has to be first, okay.

And now you have to think about what goes in the second struct. So, again, we’re having a struct in here basically because it makes sense that we group all the semaphores that have to do with the inspection and the clerks and the manager together, right? But there was nothing that would have – kind of prevented us from having them all independent, just having grouped the semaphores up there. We’re just grouping them just so it makes sense to you, okay.

So let’s look at the other struct. So we have another struct, we call it line. And this struct has all the semaphores that would handle the communication between the customer and the cashier.

Now the customer has to go to the cashier, it has to stand in line and it has to wait for its turn. So basically it has to take a number, right, which is the next place available in the line, okay? So that’s have in here something called the number. And this number – this struct again with all the fees that I’m gonna put in here is supposed to be initialized in semaphore and setup semaphores in here, okay, in the very beginning. I just kept it to the end so that it makes sense with what we’re talking about.

So the number would be initialized in the beginning to zero because this is the first place in the line, right? Nobody’s there yet, okay? Now the cashier is gonna be waiting until somebody shows up in line, right? And when somebody shows up in line this means that a cashier is requested, okay.

So we have a semaphore that the cashier is gonna wait on saying, “I’m waiting until I’m requested.” Okay? Until somebody basically shows in line. So we have a semaphore requested and this is going to be initialized to zero.

We also have another semaphore; this is a little tricky, now each customer with the cashier they have some kind of synchronization going on, right? Every customer has to go and say, “Okay, signal me when you are done with my bill. And it has to be me because I am waiting in the queue in this specific position.” Right?

So there should be some – it’s not really that we are the customers and we are waiting for you, cashier, to signal somebody of us. It’s not really somebody, it’s not any one of us and somebody will leave. We’re not caring about all the customers leaving in the end; we are really caring about every single one of them leaving in his turn. Right?

And this should give you an idea about having this kind of struct that would have a semaphore that is a rendezvous semaphore between every single customer and the cashier but this semaphore has to be for every single customer because it has to maintain the order of every single customer. Does that make sense?

So in here we will have a semaphore, let’s call it customers, and those are the customers that basically requested the cashier’s service but are waiting to be serviced, okay? So this is gonna be an array of ten semaphores, one for every single customer in turn, okay. So you come, you pick a number, you know you’re number three, then you are gonna wait on the semaphore of three. And the cashier, once he gets to your turn he’s gonna signal customer of three so that’s you – it’s your turn to leave. Make sense?

Student: Why ten customers?

Guest Instructor:Because we just assume ten customers. You could have a single number, yeah.

Student: So one point that the customer in the front of the line makes their request and that – and then he leaves and then the next person makes their request and he leaves so that the line of clerk only deals with the one – only one customer.

Guest Instructor:This is basically what we’re doing. What you’re trying to say is how can we – so let me ask you the question, okay? How can you insure that the customer – the first one in the line is the one that came in order? Because basically remember that you are not keeping track of the state. There are customers that finish and they get all their cones and they go to the cashier, right?

You don’t remember who came first. You don’t remember the order. You don’t remember which one picked the number, right? If you say the one in the beginning of the line that’s what we’re doing basically, we’re keeping track of who is at the beginning of the line.

We’re gonna say if it’s the one in place number zero and move on to one which is basically the beginning of the line. If you say that they shift, right? But if you don’t do that then you don’t know because you have like threads that finished at random times and you have no clue whatsoever which finished first. Make sense? Yes?

Student: If you had multiple semaphore waits and you do just one signal will all of them wake up or just one wake up?

Guest Instructor:Are you talking about those arrays that I’m having here? When you do a wake up you wake up the specific semaphore. You don’t wake up any one of them.

Student: But generally do you just [inaudible] waiting on a semaphore?

Guest Instructor:At least if you do wake up –

Student: At the signal.

Guest Instructor:– what happens typically – this is an OS issue, but typically what happens is when you have somebody waiting on a semaphore it’s put on a block list, right? And so you have all those threads that will be put on the block list and it depends on the scheduling issue. So it depends on whether you pick a shortest job first or you pick a priority scaling, whatever. But whoever gets to be scheduled once he finds that his semaphore was signaled then he will be able to proceed, okay?

All right. So now we have the customers and we’re still missing one thing which is very similar to the one thing that we missed before. Can anybody see it this time?

Student: Having a lock on the cashier?

Guest Instructor:Having a lock on what?

Student: On the cashier.

Guest Instructor:Having a lock on the cashier. Why do we need the lock?

Student: Because they can only deal with one customer at once.

Guest Instructor:But actually you can see that the cashier is maintaining this thing because the cashier is the one that’s gonna be waiting for requests and he’s gonna be spawning basically – he’s gonna be signaling one of those. Anybody sees any other problem?

Student: Yes, a lock for the number.

Guest Instructor:Exactly. We need a lock for the number. Basically you can think about this. Two guys go, they check what is the number. The number is zero. The other one also – and the first one gets out of the processor. The other one gets the – gets to be scheduled, checks the number, it's still zero and they both increment the number; they both get the number one, right?

So this is why, again, we need the lock on this number to make sure that this is the shared resource, this is where we have race conditions and this is what we want to secure, okay? So let me have a semaphore lock and to what is this semaphore gonna be initialized?

Student: One [inaudible]?

Guest Instructor:Yes, it's gonna be initialized to one and I think I will need to write the last few lines very quickly because we're running out of time. All right, so let me write the cashier first and then we go to this guy.

Here's what the cashier's doing, very easy. This is the cashier. It's not taking anything. And what the cashier's doing is basically for the number of customers what he's doing is he's waiting to be requested. Once he's requested he handles the bill, checks out the customer and then signals the semaphore, okay.

So you have a for loop for int I is equal to 0, I less than 10 int I plus, plus. What you want to do is semaphore wait on line. So this is line. So you wait on line until requested. Once you are requested then you know that you are basically servicing the first customer in line. So you check out some function. Check out customer I and then you signal customer I because you know he's the first one in the queue – in the line.

So you semaphore signal line dot customer of I, all right. And that's your cashier. Now let's get back to the customer. So you walk to the cashier, okay? What you want to do is you want to signal to the cashier that you request a service and wait for the cashier to handle your bill, right?

So you want to semaphore signal line dot requested and then you want to semaphore wait. Oh, I forgot something very important. I don't know my number, right? I didn't even get a number, okay? So before you request you want to get a number but before you get a number you have to lock, right, because you don't want anybody else to be getting a number at the same time.

So we want to semaphore wait on line but lock. You want to get the number so we simply an int, let's call it to replace, this is equal to line. – what did I call it? Number plus plus so that next time is the next place on the line, okay?

And then you want to signal definitely this lock because you are done using the number. And after you signal this lock, maybe right here, you want to signal to the cashier that you are requesting his service. So you signal line dot requested and then you want to wait until he handles your bill.

So basically wait on line dot customers of your place, okay? And this is your customer. I don’t think I’m missing anything. Does it make sense? Are you guys confused? This is hard. This is not an easy – it’s not that it’s – it’s not as complicated, it’s just that it’s just a lot of problems going on, a lot of synchronization issues going on.

So I would definitely suggest that you guys go and do this again, okay? You have to think about it. You have to think about the interactions between all of the players. You have to think why you need those semaphores and how you’re gonna use them. I’m gonna see you again tomorrow in this section. We’re gonna go over another example for the studying and it’s gonna be related to your assignments – to your project, okay?

Okay, have a good day.

[End of Audio]

Duration: 52 minutes