CS 240: Race Conditions in Chess Transcript

[00:00:00] Now that you know how to create threads, how to use thread pools, you know about different race conditions and you know how to protect those with um a combination of database transactions, synchronized methods, synchronized code blocks and atomic variables.

[00:00:15] Um I want to show you uh some of the race conditions in your, in your chess code, your client in your server.

Start visual description. The professor demonstrates various race conditions in the chess code, including multiple users registering concurrently with the same username, multiple users claiming the same side of the game, and joining and leaving games. The professor shows how these race conditions can occur and explains that these are not hypothetical but likely exist in the students' code. End visual description.

[00:00:22] Um I already pointed out a few, I want to show you those again and show you some others.

[00:00:27] And these are, are not hypothetical things.

[00:00:29] These are things that probably do exist in your code.

[00:00:32] Um and as I said, in a previous video, we don't expect you to resolve these.

[00:00:36] Um You don't necessarily have time for that in the class. If we had um time for a few more phases of the project, maybe this would be another phase we would add is tell you to go back and fix all of your, all of your potential race conditions.

[00:00:49] Um So this is just instructional. You don't have to do this. Although if you want to get serious about using your chess server and your chess client, you would need to do this OK. So, we've already seen some of these, we've seen that multiple users registering concurrently with the same username would, would create or

could create a, a race condition, multiple users claiming the same side of the game um is a potential race condition.

[00:01:15] And joining and leaving the games are potential race conditions.

[00:01:19] And depending on how you are doing your code, even joining and leaving different games could, could cause these race conditions.

[00:01:25] So for example, if you create the um the data structure that holds connections the first time you need it, then that that's a race condition that could span even multiple games.

[00:01:38] OK.

[00:01:39] So if we have so, so if we have the situation, but you do have the situation where multiple threads could write to the same web socket at the same time.

[00:01:50] Another um situation that's similar to that is your server could create too many database connections.

Start visual description. The professor explains a situation where multiple threads could write to the same web socket simultaneously, leading to potential race conditions. The professor also discusses the issue of creating too many database connections, which could result in running out of connections. The professor demonstrates how to prevent these scenarios using game locks and synchronized code blocks. End visual description.

[00:01:56] You could have um a situation where you have multiple users connecting and multiple database connections are getting created, you could actually run out of database connections.

[00:02:07] And so both of those scenarios can be prevented with what we call game locks or um have some lock that will lock the entire game or some um subset of the game for multiple instances.

[00:02:21] So that would be synchronized code blocks um create some game lock object that is a static variable somewhere and access that from the methods. Um And what I mean by that is have your methods have a synchronized code block in them that synchronize on that lock object, and you would be protecting all those methods from each other.

[00:02:42] Um One thing that you should be aware of is your chess client actually has two threads.

[00:02:48] So I've already pointed out that your server with your processing of web socket messages, those create separate threads, even though you didn't explicitly create them, your client has multiple threads as well because you have the main thread that's executing your code and you have your web socket code that is receiving messages from the server every time it gets a message that's processed in a separate thread.

[00:03:13] So even though you didn't write separate threads, you do have separate threads.[00:03:17] So multiple threads could access the same instance of your game object.

[00:03:22] So you have your game object that you're probably using in your client to print out your game.

[00:03:28] Well, if your web socket, uh if you get a message from your web socket and that causes something that uses your game, you have a potential race condition there.

[00:03:40] So that can happen by um you could synchronize all code that rise to the same game object, synchronize those methods.

[00:03:49] If the game object is managed within one instance of a class, or if it's multiple classes or multiple instances, you could use a synchronized code block one that you may have actually seen when usually when I teach this uh this class in person, I will ask students if they've had a situation where they printed out their game board.

[00:04:10] And every once in a while, they see a message in the middle of the board.

Start visual description. The professor describes a situation where a message could appear in the middle of the chess board due to multiple threads printing to the terminal simultaneously. The professor demonstrates how to synchronize all code that prints to the terminal to avoid this issue. The professor provides examples of synchronized methods and code blocks to ensure proper synchronization. End visual description.

[00:04:14] So they see part of the chess board and then some message that maybe says John just joined the game as user black something like that, that could appear in the middle of your chess board.

[00:04:24] And the reason for that is because as I said a minute ago, you have multiple threads.

[00:04:29] So if one thread is in the process of printing out the game and it's not done with that.

[00:04:35] And a web socket message comes in that web socket message could be printed right in the middle of your game board.

[00:04:41] Um So some of you um probably most of you won't see that, but some of you probably will see that if you play your game enough.

[00:04:48] Um So the way you can do that is synchronize all the code that prints to the terminal. So if you have all those, if you have methods being called from one

instance, maybe you have a chess client instance, a chess client class and you have one instance and everything is being printed from that one class, then probably a synchronized method would work.

[00:05:13] If you, if you could potentially be printing to your terminal from multiple classes, then you'd need a synchronized code block.

[00:05:19] So those are just a few examples, there might be a few others.

[00:05:23] Um a few other cases where you could have a race condition in your chess program, but you definitely could have those.

[00:05:29] So those are just a few examples of things that in a, in an actual production chess server and client, you would need to resolve those with the, with the techniques you've learned about concurrent programming and multi-threading.