CONCURRENT PROGRAMMING

CSC2002S PCP2

Ethan Wolff

26.08.2023

WLFETH001

INTRODUCTION

For each of the simulation rules, a paragraph is given explaining how it is enforced using concurrent programming techniques. Andre the barman was implemented by this program, as explained below.

START

In order to force all threads to wait for the start button to be pressed, a latch owned by the ClubSimulation is created and shared with every Clubgoer. The threads wait at the latch until it is released. The latch is a CountDownLatch with its count set to 1, so the main program can decrement its count when the start button is pressed. This means the latch is released and all the threads can begin, starting the program. Latches are perfect for this application.

PAUSE

An AtomicBoolean called paused, also owned by the ClubSimulation, is shared between all Clubgoer threads. Between each action, patrons check this boolean and if it is true, wait for it to become false again. This means that the main program can simply toggle this boolean to pause and unpause the threads, ensuring that the notifyAll method is called to free the paused threads. This is clearly appropriate as we need a global-adjacent boolean that all threads can access without causing issues, which is the perfect use-case for an AtomicBoolean.

EXCLUSIVE GRIDBLOCKS

Two Clubgoers should not be able to enter the same gridblock at the same time, but without synchronization, a bad interleaving of two Clubgoers calling ClubGrid.enterClub() wil cause this. This is avoided by synchronizing the get method of GirdBlock, which means that inside of ClubGrid.enterClub(), the first ClubGoer to call it will get the lock, and set the GridBlock as occupied, which means when the lock is released and the second ClubGoer attempts to enter, it will see that occupied is true and not move.

EXCLUSIVE ENTRANCE AND EXIT

The exit is already exclusive because it is not entered differently from any other GridBlock, but the entrance has special entrances. To avoid problems, the entrance is locked by a single Clubgoer waiting to enter, and they wait until the entrance is not occupied to enter. We notify them to check this when a patron leaves the entrance square, which we can check for within the ClubGrid's move method. Once the Clubfoer moves, the waiting ClubGoer is notified, they see that the entrance is not occupied and they enter, assuming the limit of Clubgoers has not been reached.

PATRON LIMIT

In a similar way as above, synchronization is used to ensure that the limit of patrons is never surpassed. After a Clubgoer ensures that the entrance is free, they lock the PeopleCounter and simply wait until the count has gone below the limit so they can enter. The PeopleCounter is notified whenever a Clubgoer leaves the club so the waiting Clubgoer will always know when it is alright to enter.

LIVENESS AND DEADLOCKS

By the code provided in the skeleton, all the patrons move simultaneously as they are separate threads, which is maintained in the updated code. Liveness refers to a concurrent application's ability to execute in a reasonable amount of time, with a reasonable amount of time being dependent on the problem. This primary threat to liveness is deadlocks. This program avoids deadlocks by ensuring that there is never a situation where an object locks on another object and then requires another lock as well, as this is when a deadlock can take place. A deadlock would take place if another object has locked the second resource and is waiting for the first object to release the first resource. By not locking multiple resources, this program sidesteps this problem entirely. For example, when the entrance GridBlock or PeopleCounter are locked, nothing else is required or locked. With a large enough number of patrons, the club can become blocked, but this reflects reality, and solving this is out of the scope of this project.

ANDRE THE BARMAN

Andre the barman is implemented as another thread, similarly to how Clubgoers are implemented. He uses the same CountDownLatch start and AtomicBoolean paused as the Clubgoers to ensure that he respects the start and pauses of the program. He moves up by calling ClubGrid.move just like the Clubgoers, but moves up and down the bar in a straight line. When he reaches the end of the club, he turns around, serving patrons as he passes opposite them.

Clubgoers need to not leave the bar until they are served once they are there, so when they arrive at the bar they lock on to their current GridBlock, waiting until they are released. When Andre reaches the square across the bar from them, he serves them a drink by notifying their block and thus releasing them to return to the bar.