Implementation decisions

In the main class there is a debug mode to show the threads locking variables. This makes the deadlock easy to find.

We used synchronized, instead of semaphores to prevent full parallelism of the threads because it is suggested to use this method when reserving/acquiring/locking the variables.

We use an object array as our control variables since they do not need to do anything else.

We chose to lock control variable ri-1 before control variable ri because it should not matter which one is locked first.

Challenges

We encountered several problems while trying to complete this assignment. Here’s a list of the progression the challenges we faced.

* Having DoAction() in the synchronized block caused deadlock.
  + We moved out of the synchronized block and it seemed to delay the deadlock.
* Having acquire and release both as synchronized methods caused deadlock.
  + We had acquireLock() and releaseLock() both as synchronized methods but did not realize that if one of them was running, it locked all of the code in the class instead of just that method.
* Tried semaphores instead of synchronized.
  + We decided to try to use semaphores instead of synchronized and the program ran for longer but still ended up resulting in a deadlock.
* Figured out how to implement synchronized correctly.
  + We then tried to implement synchronized correctly and the program worked correctly and eventually ran into the same problem as using semaphores, which was a deadlock.
* DoAction() with each lock and acquire.
  + Reading over the assignment again, we saw that we were only running DoAction() twice per thread instead of four times. Once we added the other two instances, it caused a deadlock to happen in less time than before.

Issues

We found a big issue within the code no matter what we tried to change. The issue is that we are trying to lock two variables at the same time. The DoAction() causes there to be a delay in between the locking of the first and second variable. This delay allows many threads to lock their first variable but not the second, and eventually the threads have the first variables locked and the remaining threads need to access those variables, causing a deadlock.

Compilation Instructions