COMP2240 Operating Systems, Semester 2 2022

Assignment 2: Testing Semaphore and Monitor-based Simulations

by Olivia Favos (3188124), October 2022

1 Introduction

Assignment 2 involved three tasks, where the aim was to solve different problems using semaphores and monitors to design simulations which prevented starvation and deadlock. The purpose of this report is to outline my approach to testing the algorithms used, demonstrate how I constructed the simulations to meet project requirements, and commenting on noteworthy issues encountered throughout development.

2 Testing Overview

To test P1, the debugger in the IntelliJ IDE was used. Each time a change was made to the file, I re-compiled and re-ran the program to ensure that new code did not break the program. Testing P2 involved the same methodology. P3 has not been developed, and therefore did not involve any testing.

3 Enforcing Mutual Exclusion

Throughout Assignment 2, all three tasks required the implementation of different algorithms to enforce mutual exclusion. In P1, threads simulated multiple/concurrent Farmers using a Bridge, represented by a binary semaphore. This semaphore allowed only one Farmer to cross the Bridge at a time, demonstrating mutual exclusion. P2 introduced a counting semaphore which represented the number of seats in the Parlour. The third task, P3, used a monitor to design an algorithm which enforced mutual exclusion. In the way I had intended to design P2, P3 would have only required changing a small amount of code to run according to specifications.

4 Preventing Deadlock and Starvation

In P1, to provide an algorithm which was starvation-free, semaphores were used to ensure that waiting Farmers remained locked at their location and could not cross the bridge to reach their destination until the currently crossing Farmer had finished. P2 presented a new challenge in ensuring that the solution produced was starvation free due to the manager's special rule, which states that when all seats in the ice-cream parlour are occupied, all arriving customers must wait for the entire party to leave before they can be seated.

5 Conclusion

Overall, I found that it was simplest to enforce mutual exclusion using a semaphore in P1. I was ultimately unable to complete P2 or P3, due to a null pointer exception error which I was not able to resolve before the time of submission.