CS444 Concurrency Assignment 3

Mark Bereza and Morgan Patch

Problem 1

Design

For Problem 1, the desired functionality was achieved using three different locking mechanisms. First, a semaphore is used to keep track of how many threads have access to the shared resource. Each time a thread ”acquires” the resouce, the semaphore counter is decremented by 1. Similarly, whenever a thread is finished with said resource, the semaphore is incremented by 1. When the semaphore's value reaches zero, threads attempting to access the resource will block. The last thread to successfully access the thread will flip a global flag, the second locking mechanism, that will prevent any threads for accessing the resource or touching the semaphore. This flag is flipped back once all three threads that had access to the resource release it, thus bringing the semaphore's counter back to 3. The final locking mechanism was a mutex used to ensure that the semaphore value was not altered between when the value was queueried and when it gets printed. Thus all threads attempting to increment the semaphore, decrement the semaphore, or query the semaphore's value must first acquire this lock. The lock is released immediately after the increment/decrement/query operation completes.

Running the Program

The program can be ran by first running ”make” and the running the problem1 program produced by make. If a numerical argument is provided to the program, that many threads will be spawned. If no arguments are provided, the program will spawn a default five threads. To terminate the program, simply send it an interrupt signal using Ctrl+C.

Ascertaining Correctness

The solution to Problem 1 was determined to be working correctly by simply inserting print statements whenever the semaphore's state changes (indicating the new semaphore value and the thread responsible for the change) and running the program. Just looking at the printed output will show that no more than three threads ever have access to the shared resource simultaneously and as soon as the third thread accesses it, the resource is locked until all three threads release it. Outside of this locked state, threads can freely access the resource so long as the semaphore's value is above zero. Thus, the solution achieves the functionality outlined in the problem description.