Justin Sprouse and Ethan Wilkes

COMP 3500

Homework 3

March 28, 2022

Homework 3 Writeup

Part 2.

1. 3 sample outputs from part 2’s implementation

Group 1 (0..9): 11

Group 2 (10..19): 5

Group 3 (20..29): 12

Group 4 (30..39): 9

Group 5 (40..49): 11

Group 6 (50..59): 10

Group 7 (60..69): 13

Group 8 (70..79): 11

Group 9 (80..89): 7

Group 10 (90..99): 11

Group 1 (0..9): 11

Group 2 (10..19): 5

Group 3 (20..29): 12

Group 4 (30..39): 9

Group 5 (40..49): 11

Group 6 (50..59): 10

Group 7 (60..69): 13

Group 8 (70..79): 11

Group 9 (80..89): 7

Group 10 (90..99): 10

Group 1 (0..9): 11

Group 2 (10..19): 5

Group 3 (20..29): 12

Group 4 (30..39): 9

Group 5 (40..49): 11

Group 6 (50..59): 10

Group 7 (60..69): 12

Group 8 (70..79): 11

Group 9 (80..89): 7

Group 10 (90..99): 10

1. Differences in runs

Firstly, we know that part 2 could theoretically produce inconsistent results. This is because in this current implementation, **there are no process / thread synchronization mechanisms in place**. **This could lead to two threads trying to access the same variable at the same time, leading to an incorrect value.**

For example, if thread A and thread B both read in a number that belongs in group 1, the steps could be as follows:

1: Thread A reads in a number and determines it belongs in group 1. It then accesses the count variable for group 1 and sees it is 5.

2: Thread B simultaneously reads in a number and determines it belongs in group 1. It then accesses the count variables for group 1 sees it is 5.

3: Both threads try to increment the count value by adding one, in both cases coming up with 6. However, we can see this should have been seven.

This is one example of how this process could have inconsistent results.

**There are slight differences in our outputs between runs**; however, results tend to be the same for the most part. These slight differences are due to the reasons listed above.

Part 3

1. Sample Output

Group 1 (0..9): 11

Group 2 (10..19): 5

Group 3 (20..29): 12

Group 4 (30..39): 9

Group 5 (40..49): 11

Group 6 (50..59): 10

Group 7 (60..69): 13

Group 8 (70..79): 11

Group 9 (80..89): 7

Group 10 (90..99): 11

1. This output is **identical to our output from part 1**. Our output for Part-3 and part-2 were **similar, but they had the occasional difference** for the reasons listed in part-2’s answer. Basically, part 2 and part 3 had different answers because the lack of a mutex in part 2 let the threads access the count variables at the same time, creating inconsistent results. These inconsistencies were worked out with part-3’s implementation due to the use of mutexes.

The mutex synchronization approach spawns ten “**locks**”, each correlating to a count variable. When a thread needs to access a count variable for a certain group, it tries to lock the corresponding mutex. If it is able to lock the mutex, it knows that that count variable is available, and proceeds to increment the count, unlocking the mutex after it has completed. However, if it is unable to lock the mutex, it knows that the corresponding count variable is currently occupied and waits until the mutex has been unlocked. Once the mutex is unlocked, the thread knows the resource it needs is currently available, locks the mutex, and begins performing its operations. In this way, no count variable is able to be accessed by multiple threads at once