CSC 33200 Lab Jonathan So

**Task 5 Report**

**Step 1**

When we run the original code, we have a race condition between the dad and son processes. The threads are all accessing and updating the shared data simultaneously; the dad and son processes are all updating the balance value. This causes there to be no order resulting in invalid executions and bugs. In order to fix this problem, we need to add locking. We do this through using a semaphore. Using the semaphore operation, P, we lock the critical section when a process is updating the shared data. This way it will allow mutual exclusion; other threads can’t access and change the data. After a process is done updating the data, we use semaphore operation, V, to exit the lock, so other threads can update the shared data. In the case of our code, we use P and V to lock all our processes’ (dad, son 1 and son 2) critical sections.

**Step 2**

In order to keep track of the number of times a process is interested in entering a critical section, I created three additional files (\*fp5, \*fp6, \*fp7) to keep track: dad\_time, son1\_time and son2\_time. I used counters for each process and in each process, the other 2 processes’ counters are incremented if they show interest in entering the critical section. After running the code, the dad process showed interest in entering the critical section 22 times, while both the sons’ processes showed interest in entering the critical section 16 times.

**Note**: In my code, changes I made to fix synchronization and the wait time are in between the

“////////////////////////////////////////////////////////////////////////////////////////////////////”