**Discussion:**

My program simulates a post office, with 50 customers and 3 workers. I followed the criteria of the instructions to as much detail as I could, using semaphores as the only means of synchronization, and having the required output be as correct as I could make it. I have the customers created first, then they interact with worker threads as they are created.

**Difficulties:**

There were three main difficulties I encountered while making this program. The first was simple syntactical errors as I was learning how to create threads and manage semaphores. Thanks to many online sources, such as Linux.die.net, and a huge amount of examples through sites like stack overflow, I was able to make the necessary corrections.

I had also hilariously implemented a stack instead of a queue by habit, producing starvation for many of the first created threads. This was certainly easy to correct, but only after I was able to identify the problem.

The second difficulty was getting the order of the semaphores correct conceptually to not produce deadlocking. I decided, rather foolishly, to begin coding without thoroughly examining my psudocode as heavily as I should have. My design in the beginning had the order of some signal interactions backwards, producing deadlocks. This was eventually remedied.

The last, and largest issue I had was finding out why my threads were terminating prematurely. All of the threads were being created properly, but they would stop after I processed about 8 customers. I thought there was some kind of timeout issue going on, where threads were, perhaps, waiting too long on a semaphore to unlock, and terminated. It had turned out that whenever the main function had completed, and it terminated, all other threads terminate with it, returning me to the bash shell. After much research, I experimented with a wait timer at the end of main and discovered that this was the issue. Then it was trivial to add the join function, keeping main alive until all the customer threads were complete.

**Results:**

I am very happy with the output, and how the wait times as implemented from the task table, helped make the program’s output flow. There are occasionally concurrency issues with printing, as I had not implemented the means of mutual exclusion when printing. I decided to add the semaphore MUTEX3 to make this happen. This made the results much cleaner, wince multiple threads will now not print at the same time.

**Lessons learned:**

I had no idea that threads were this powerful. Especially since semaphores make concurrency much easier. I can see how interactions between items can be better simulated, and controlled, all while being easy to write and understand what is happening.