Corey Robins

111185399

Design Document

For this project, I followed the general implementation that was provided in the skeleton code. In terms of data structures, I used the variable “sim\_pid” to keep track of the PID of the main process, and a “sequence” variable (of type sequence) to generate the order numbers for each Eater. I also used a blockingqueue (“eaters”) for the Eater to tell the Cooks that it has placed an order, and then grill\_queue, frier\_queue, and soda\_queue for the Cooks to communicate with the respective Machine that it should start cooking the Food. I then used 3 more semaphores (one for each Machine), and each semaphore has as many permits as the specified capacity of each Machine. I also kept the semaphore for the Eaters, for which an Eater can only start eating if it obtains a permit (equal to the number of tables available).

In terms of actual implementation, I will discuss a few things that I think may need some explaining, and the rest is made clear through the comments in the code itself. For the Machines, I just started up 3 Machine processes (one for each), and used semaphores to monitor the capacity. The Cook put its own PID on the queues, and once the Machine finished cooking the specified Food, it sent the message over to the Cook.

For the Cooks, I had each Cook block until an Eater was available to process. Once one was available, it sent the message over to each Machine process to cook the 3 foods in the order, and then waited to receive a message from the Machine that it had finished the Food. Once it received 3 messages, then it sends a message back to the Eater that it has finished the order. The “eaters” blockingqueue takes tuples of {Eater\_PID, Order Number, List of Foods for Order}.

Lastly, the Eaters block until there is a permit available on the “restaurants” semaphore, signaling that a table is available. It then places itself onto the “eaters” blockingqueue and waits to receive a message from a Cook that it has completed the Eater’s order.

In terms of testing, I just ran the provided test file with as many different cases as I could think of. I tried with more tables than eaters, higher machine capacity than cooks, fewer cooks than eaters, and other cases in which an error might occur. Since the code wasn’t too long, I also manually stepped through the code after writing it to make sure that it logically made sense and was efficient. Additionally, even if the execution “passed” the validation at the end, I tried to manually inspect the output (in smaller test cases) to make sure that the output was actually correct and there were no small bugs that the validator was missing.