Q2 Report
Aum Alok Khatlawala
2020113008

Algorithm Used and Implementation Details:

After receiving the input in the given format, first I assigned the values for the different variables to the structs of each entity in the problem (pizzas, ingredients, chefs, customers and their orders).

After this, I sorted the customer order list similarly to the way it was done in the first question.

After this, I create two semaphores called chef_sem and oven_sem respectively and I initialise chef_sem to 0 and oven_sem to o (number of ovens specified in the input).

After this, I create threads for the customers and chefs as threads in a way identical to the way it was done in the first question.

In the thread function for chefs, I implement the following logic:

1. Sleep till arrival time.
2. Perform sem_post to chef_sem when the chef arrives.
3. Sleep till exit time.
4. Perform sem_wait to chef_sem when the chef exits.

In the thread function for customers, I implement the following logic:

1. Sleep till arrival time.
2. Print customer and order details as per the directives of the question document.

3. For each pizza order, wait for a chef to be assigned. If the chef doesn't leave before the pizza can be prepared, reject the order. If not, check if the number of ingredients is sufficient for making the pizzas. If the number of ingredients is not sufficient, reject the order for that particular pizza.
4. If the number of ingredients is sufficient, accept the order and assign it to the chef. After the chef arranges the pizza toppings on the pizza, wait for an oven to be assigned to the pizza and after the oven bakes the pizza, give it to the customer in the drive-thru zone. Handle the sem_wait and sem_post of oven_sem accordingly.

After the threads finish execution, we join the threads for chefs and customers using pthread_join and after this, we destroy the semaphores for ovens and chefs using sem_destroy.

Follow-Up Questions:

1. The pick-up spot now has a stipulated amount of pizzas it can hold. If the pizzas are full, chefs route the pizzas to secondary storage. How would you handle such a situation?
   Ans: This situation can be handled using the FCFS approach. I would create a time variable for when each order was placed by each customer. Depending on the customer index and the time they placed their orders, I would keep the stipulated amount of pizzas in the pick-up spot using a queue and would send the other pizzas to the secondary storage so that the customers who have placed their orders first can get their orders first.

2. Each incomplete order affects the ratings of the restaurant. Given the past histories of orders, how would you re-design your simulation to have lesser incomplete orders? Note that the rating of the restaurant is not affected if the order is rejected instantaneously on arrival.

Ans: I would redesign the simulation in such a way that every time an order is placed for n pizzas, I do a precomputation of the possibility of rejecting even a single pizza from the order. The code would be nearly identical to the one written but if there is even a single pizza that gets rejected from the order, I set a flag to 1 and reject the order when the order is placed. This computation might take a bit more time to simulate but it is a sure shot method to avoid the rating from going down.

3. Ingredients can be replenished by calling the nearest supermarket. How would your drive-thru rejection/acceptance change based on this?
   Ans: There wouldn't be too much of a change to the code. I would just add a condition that if at any instant, there is a replenishment of ingredients occurring by calling the nearest supermarket, then I reset the values of ingredients in the ingredients list based on the ID of the ingredient. I might need to create a thread for this if the time for the replenishment of ingredients is mentioned in the input. The thread will function by sleeping till the replenishment time and then resetting the value of the ingredients. This can easily be changed to handle specific amounts of the ingredients to be replenished instead of resetting them to their original values.