The structure of how the system works.

1. Read all the orders to a queue. Launch a thread that will count timestamps and bring orders to the system at their respective timestamps.
2. When the issue time of order and current time in the system are the same, the order is processed. For each pizza in an order, find oven with least time to complete its orders among others and add pizza to this oven’s queue. Increase queue\_total\_time attribute’s value by the time it takes to cook a certain type of pizza. This variable will help us to promptly compare ovens.
3. For every oven we start a thread. Thread monitors whether there are any pizzas to be cooked and if yes, it cooks it (sleep function). When done, it pops a pizza from queue. For order object, increment the attributes large\_pizzas\_ready/small\_pizzas\_ready depending on the size of cooked pizza. Check if order completed here, by checking whether the attribute values (large\_pizzas\_needed == large\_pizzas\_ready) and (small\_pizzas\_needed == small\_pizzas\_ready).
   1. If yes, order is completed. We need to make customers seated. If all the tables are busy, the order is put into a queue, that will be managed by a thread. We run a code to find a table that has minimum difference (left\_seats – number\_of\_customers). This thread periodicaly checks availability of tables and assigns customers to them. Make customers seated on that table.
4. We start a thread that will mimic customers eating. To compute the time it will take them to eat, we will use this formula: total\_time\_for\_customers\_to\_finish\_their\_pizzas = time\_to\_eat\_large\_pizza\_by\_one\_person / number\_of\_customers \* number\_of\_large\_pizzas\_in\_order + time\_to\_eat\_small\_pizza\_by\_one\_person / number\_of\_customers \* number\_of\_small\_pizzas\_in\_order. After this time passes, order is closed, table seats are released.