**Synchronization approaches:**

// Yan Wu

// enter shop

1. Simulation starts, create a queue of current customers;
2. customer objects created (enter shop);
3. require the lock of current customers queue, if the size equals the capacity of the shop, this customer thread wait, and release the lock;
4. if any customer leaves the shop, require the lock of customer queue, remove the left customer, and notify all waiting customer threads

// place order

// plan to create two list of cooks, one for busy cook, one for idle cook

1. require the idle cook queue, if no cook is available now, wait the current customer thread
2. if any order finished, require the lock of the busy cook queue, do the change,

release the lock;

1. require the idle queue, add the cook, notify all waiting customer threads, release the lock;
2. once a customer’s order status is submitted, he waits;

// process the order

1. iterate the order, require the lock of machine, if it is full, the current cook wait, release the lock;
2. When any machine finishes a food, require the lock of the machine, change the capacity, notify all, release the lock.;
3. The machine with finished food, notify the waiting cook, to require the lock of the cook, change its order status, release the lock;
4. Once a cook finished his order, he requires the lock of the customer, change the order status, notify him, release the lock.
5. Once a customer finishes his meal, he requires the lock of customer queue, remove him self, notify all, release the lock.

Some additional properties to help this work:

Order class with list of food, and status of the order

The machine class: timer for each food processing;

Simulation: queue of idle cooks, queue of busy cooks, queue of customers in the shop