Report - Concurrency

Cafe Sim

Basic Implementation Idea

States are defined for the barista (`AVAILABLE` and `OCCUPIED`) as well as for the customer's order (`OUTSIDE`, `ORDERED`, `PREPARING`, `LEFT`). Each customer has executes in its own thread and each order as well. Each order has four semaphores to behave as conditional signal variables - `orderPlacedSignal, orderReceivedSignal, startPrepSignal,endPrepSignal`. Common locks are also maintained for updating barista, customer or order details - `baristaLock, customerLock, orderLock` as well as a `barista_sem` which waits till some barista is available.

The `simulateCustomer()` function deals with everything for a customer alone like waiting for arrival time, checking his tolerance time and exiting status. `sem_timedwait()` has been used to check if either barista gets assigned and order gets prepared before the tolerance time runs out.

The `simulateOrder()` thread deals with the orders, where it uses the four conditional signals to communicate its status with the customer and inform about the barista assigned and order preparation status. It gets the lowest indexed customer and barista and then prepares the order if customer has not left yet. Order preparation just involves sleeping for the preparation time.

Question 1

Waiting Time: Calculate the average time a customer spends waiting for their coffee. Determine if this is more than (and how much) if the cafe had infinite baristas. Coffee prep time is not part of waiting time.

Answer 1

The wait time is calculated as (from the doubts doc) -

```
watiTime = timeSpentInCafe - prepTimeOfCoffee // if customer gets coffee
watiTime = timeSpentInCafe // if customer does not get order
```

```
// (prep time is 0 as customer does not get order)
```

If the cafe had infinite baristas, each customer would get a barista assigned to them immediately when they enter. So the only significant waitTime would be considered when the order placed by the customer takes more time to prepare than the customer's tolerance time.

In contrast to finite baristas, infinite baristas would have minimal wait time (1 sec for barista to start preparation of order) for most customers which reduces the average wait time by a significant amount.

In the code, every time the customer leaves (based on if they leave with or without the order), the wait time is calculated for the customer based on the above formula.

Question 2

Coffee Wastage: Determine how many coffees are wasted. Print the number of coffees wasted after the simulation ends.

Answer 2

A coffee is considered wasted when the customer leaves before the order could be started preparing by any barista as all baristas were occupied in some other order. We can reduce the coffee wastage by signalling the customer (based on his tolerance) in case the coffee cannot be prepared in enough time whenever a barista gets free, and thus not preparing the order.

In the code, coffeeWasted is incremented only when the `startPrep` signal was given for the order but `endPrep` signal could not be received before the tolerance time ran out.

Ice Cream Parlour Sim

Basic Implementation Idea

States are defined for the machines (`BLOCKED`, `AVAILABLE` and `OCCUPIED`) as well as for the customer's order (`OUTSIDE`, `ORDERED`, `PREPARING`, `COMPLETED`, `NO_MACHINES`, `NO_TOPPINGS`). Each customer has executes in its own thread and each barista as well. Each order has four semaphores to behave as conditional signal variables - `orderPlacedSignal,`

`orderReceivedSignal, startPrepSignal,endPrepSignal`. Common locks are also maintained for updating machine, customer or order details - `machineLock, customerLock, ordersLock` as well as a `machines_sem` which waits till some machine is available. A `toppingsLock` has also been used to prevent multiple machines accessing the toppings to cause a race condition.

The `simulateCustomer()` function deals with everything for a customer alone like waiting for arrival time, checking his order status and exiting status. If a order cannot be prepared, flags are set for no toppings or no machines. In case of no toppings, all other orders are cancelled and the customer leaves.

The `simulateMachine()` thread deals with the orders, where it uses the four conditional signals to communicate its status with the customer and inform about the machine assigned and order preparation status. It gets the lowest indexed order possible by and checks if the ideal machine to be used for this order is the current machine or not. If not, it rescans for available orders it can take up before its closing time and then prepares the order if customer has not left yet. Order preparation just involves sleeping for the preparation time and reducing the toppings. Check for toppings availability happens whenever a machine is assigned and this in-turn signals all other orders to stop for the customer, unless the order was already being prepared.

Assumptions

- If machine completes preparing some order at t, then it can take up the next order at t+1
- If all limited toppings are over, then the parlour closes after sending all waiting customers with partial orders.
- Toppings check is done once when the customer places the order and then again when a machine gets assigned to a order by the customer. When toppings shortage occurs all non preparing orders are cancelled and the customer leaves with a partial order.
- Customer waits till parlour closes if no machines are able to satisfy the order of the customer.

Question 1

Minimising Incomplete Orders: Describe your approach to redesign the simulation to minimise incomplete orders, given that incomplete orders can impact the parlour's reputation. Note that the parlour's reputation is unaffected if orders are instantly rejected due to topping shortages.

Answer 1

Incomplete orders are the orders of the customer that have been partially completed due to either machine unavailability or topping shortage. An ideal way to handle this would be to "reserve" in some way the machines and toppings for the order of a customer when its placed. Doing this, we can instantly check if the new arriving order of the customer can be completed or not. If it can, then we reserve ingredients and machines time for this order, ensuring that the customer never leaves with an partial order. If it cannot, then the customer is instant rejected that does not affect the parlour's reputation.

Toppings reservation is easy as we can just keep aside the toppings for the particular order and reduce the globally available toppings by this amount, so no other order interferes with the toppings for this order. However reserving machine times is slightly trickier as we would need to keep track of expected finishing time for a machine (if it is engaged in some other order) as well as ensure there is sufficient time to prepare all orders of a customer.

Question 2

Ingredient Replenishment: Assuming ingredients can be replenished by contacting the nearest supplier, outline how you would adjust the parlour order acceptance/rejection process based on ingredient availability.

Answer 2

When ingredients can be replenished by a supplier the usage of toppings turns into a standard producer-consumer problem. We would not reject the order when the ingredients are not available, but instead have the customer wait till some some supplier supplies the toppings that can be used for the customers order. The only possibility of the customer leaving partially is if the machines all close leading to the parlour's closing. Assuming that toppings are supplied regularly, and machines have enough time, the entire order can be prepared and thus the customer is not rejected.

Question 3

Unserviced Orders: Suggest ways by which we can avoid/minimise the number of unserviced orders or customers having to wait until the parlour closes.

Answer 3

To minimise this we would again need to keep some sort of reservation system where the machine times and toppings are reserved for a order and thus the customer either directly leaves on entering (if order cannot be fulfilled) or is entirely serviced.

In case of machines the hard part would be to decide which machines to reserve for a order when multiple machines can do so. We would need to implement some sort of policy for this which would affect the future orders. We would need to let go of the constraint of having the lower idx machine prepare the order as some other order might be better for minimising future orders.