TDT4186 Operating Systems

P1

SushiBar

Markus Andersen
markusa@stud.ntnu.no

March 5, 2018

Table of Contents

1 Implementation

In this assignment we where given source code for *SushiBar*. We had to implement threads, and take care of resource management of these threads. It is part of the producer/consumer problem.

SushiBar
In this implementation of *SushiBar*, the main thread will first create the objects needed such as the *Clock* and *WaitingArea*. Then it will create threads for the *Door* and the *Waitresses* before it starts the threads with *start()*. The main thread adds the threads to a *LinkedList*, which I used to later *join()* the threads. This was to pause the main thread for the other threads to finish, before the program would print the statistics.

Door
The class *Door* will run a while loop on *SushiBar.isOpen* which created new customers and added these to the WaitingArea with *waitingArea.enter(customer)*. The *Door* would then sleep for a set time *SushiBar.doorWait*, unless the *WaitingArea* would be full and then make the thread *wait()*. It would then resume once a customer was fetched by a *Waitress* and notified with *notify()*.

WaitingArea
The class *WaitingArea* has a LinkedList with all the customers added through the *enter(customer)* function. This function will make the thread *wait()* if there is no more room in the queue. After the thread is resumed it will add the customer to the queue with *queue.add(customer)* and *notify()* a *Waitress*. The function *next()* will make the *Waitress* thread *wait()* if there is no customers in the queue. It would then resume once a customer is added to the queue by the *Door*, it will *notify()* the *Door* that there is more room before returning the next customer to the *Waitress*. We can unlock the *Door* before the room is free because the *Door* will sleep a set time before creating a new *Customer*.

Waitress
The class *Waitress* will run a while loop on *SushiBar.isOpen* and check if the queue is not empty. The *Waitress* will then sleep for a set time before taking the customers order with *customer.order()*. The thread will then wait for the customer to finish before fetching a new customer.

Customer
The class *Customer* will run *SecureRandom* on a number between 1 and *SushiBar.maxOrder* and set this as the *customerOrders*. Then *SecureRandom* is used to determine how many of these orders are *customerBarOrders* and *customerTakeawayOrders* and add them to the *SyncronizedIntegers* stored in *SushiBar*. The *Customer* will then wait for a set time *SushiBar.customerWait* before finishing and making the thread *Waitress* available to fetch another customer.

2 Questions

2.a Q: What are the functionality of *wait()*, *notify()* and *notifyAll()* and what is the difference between *notify()* and *notifyAll()*?

The functionality of *wait()* is to make a thread wait for available resources which can be notified by *notify()* or *notifyAll()*. The difference between *notify()* and *notifyAll()* is that when you only want to unlock one thread you would use *notify()*, but if multiple threads could do useful work you would use *notifyAll()*. In our case we want to save the resources, thus using *notify()* is the proper way.

2.b Q: Which variables are shared variables and what is your solution to manage them?

The shared variables are *customerCounter*, *servedOrders*, *takeawayOrders* and *totalOrders*. These variables are managed through the class *SyncronizedInteger* and stored in the main class *SushiBar*.

2.c Q: Which method or thread will report the final statistics and how will it recognize the proper time for writing these statistics?

The main thread will wait for the waitress threads to finish with the thread function *join()*. After these threads are done the main thread will print the statistics.