Operating Systems 2 - Assignment Semaphores - 2021

Learning objective

In this assignment you learn how to write a concurrent application in C by means of semaphores.

Case: supermarket checkout

In this made-up scenario we will model the behaviour how customers can pay and checkout their groceries in a supermarket. In the old days it was simple, you only had one cashier, only one queue, nothing to do but wait your turn. Nowadays you have the choice between multiple cashiers or do it yourself with a handheld scanner. Of course, with a handheld scanner, you might still have to wait for a checkout terminal to become available, but it is quicker than waiting in line by a cashier and putting your groceries on the moving belt (and putting it back in your cart again.

To be able to run everything smoothly, assume the following points: (note: all the times are deliberately chosen short to enable easier testing, but may/can/should be changed during testing)

- Every 2 seconds a customer arrives and takes a cart to go shopping, there are 20 carts, if no cart is available, the customer goes back home empty handed.
- About half the customers take a handheld scanner with them, there are 10 handheld scanners, if none are available, the customer does the shopping the normal way.
- Shopping (and if applicable scanning with the handheld scanner) takes around 5 seconds.
- Customers without a handheld scanner go to one of three cashiers, and if there are already customers waiting in line for a cashier, they choose the cashier with the least (or none) customers in line and join the end of that queue. When it is their turn, it takes 5 seconds to register all the items and pay.
- Customers with a handheld scanner go to one of two checkout terminals, if none are available, they wait in one queue, first customer in the queue takes the first terminal which becomes available. Registering the handheld scanner and paying only takes 2 seconds, after that the handheld scanner becomes immediately available for new customers.
- After paying, every customer takes 2 seconds to bring the groceries to their bicycle or car and return the cart, which then becomes available for new customers, and then they go back home.

Task

Implement a simulation of the above system in C, you may use your own laptop and C IDE to program and test it, but it should be possible to port it to, and run on, a Raspberry Pi. Use only semaphores for synchronisation and the correct functioning.

Test your solution by changing the time frequencies: how fast people enter, how long they shop, how long the registering and paying takes, etc. Also test with changing the number of carts, handheld scanners, cashiers and checkout terminals. In that way you can force certain behaviours.

Show (and discuss in your report) that people have to leave if no cart is available, show how they choose a cashier wisely, show that a handheld scanner is quicker, etc. Use the results to discuss and 'prove' that your solution works for longer periods of time.

Hint: Thread synchronisation by means of Semaphores always uses shared memory, to share information between threads. So, find out first which information you probably have to keep track of to simulate the case above.

Develop your solution by determining:

- which activities (threads) there are in your system,
- what an activity actually does,
- where activities have to wait for each other (synchronisation)
- which data you need to let the system run correctly

Additional features (not obligatory to pass)

For those of you who want a bigger challenge (and consequently a possible higher grade), improve your solution by adding some or all additional guidelines below:

1. Implement randomisation, that is, vary all the times: how fast customers enter (e.g. between 1 and 5 seconds), how long they shop (e.g. between 2 and 20 seconds), the time it takes to pay at a cashier or terminal, etc.
2. Every once in a while, a customer with a handheld scanner is randomly checked to see if they scanned all groceries. Implement that you have a 25% chance of being checked. Of course, this takes significantly longer then.
3. Implement manual checkout. Sometimes a customer would have liked a handheld scanner, but they were all out, or if there are long queues at the cashiers and none at the checkout terminals, a customer (without a handheld scanner) can choose not to go to a cashier's queue but instead do a manual checkout at one of the checkout terminals where all groceries are registered and paid, this will take the same time as a checkout at a cashier. Of course, theses customers also have a 25% chance of being checked!

To submit at blackboard:

1. A report in which you describe the approach of your problem. Clearly specify the threads you need, the life cycle of the threads and the way in which the threads communicate/synchronise with each other. Also specify the way how you tested your program, so that you can make it plausible in a convincing way that your solution is a correct implementation of the case (this means of course also that you should test your program in a convincing way).
2. An ***exported*** zip file with the name Surname1Surname2Semaphore (in which the surnames are of course the surnames of the group members).
3. Deadline: see blackboard.