OS Mini Project Report

Adithya Sunilkumar
IMT2021068

Introduction: The goal of the project is to simulate an online shopping environment with admins who can modify the product stocks, and the users who can purchase products. This is done using sockets in the C programming language. The code is divided into three files.

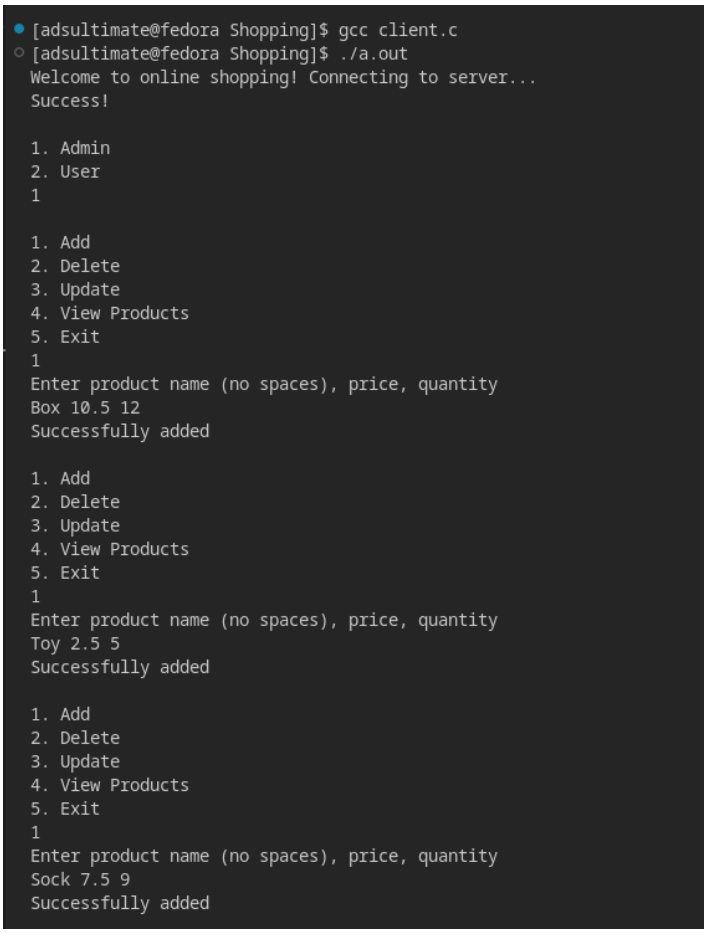
Server: Stores the products available in a global array called data. Multi-threading is used to allow many clients to connect to the server and run concurrently. The server is responsible for listening to client requests and modifying the data appropriately. Binary semaphores are used to lock and unlock each element of the array as required.

Client: A person can connect to the server through the client program. The client provides options to run as either an admin or user, both of which have different functionality.

Defn: A header file which stores common constants such as maximum size and product struct.

Running the program:

To test the program, one would first execute the server code in the backend and then proceed to run one or many client programs. Given below is an example of how an admin would run a client program, by adding three example items to the stock.



Next we can see how to view the products and modify them.
The products are always given in form ID | Name | Price | Quantity

```
1. Add
2. Delete
3. Update
4. View Products
5. Exit
4
PRODUCTS:
0       Box     10.50   12
1       Toy     2.50    5
2       Sock    7.50    9

1. Add
2. Delete
3. Update
4. View Products
5. Exit
3
Enter product id, new price, new quantity
1 3.5 6
Successfully updated

1. Add
2. Delete
3. Update
4. View Products
5. Exit
4
PRODUCTS:
0       Box     10.50   12
1       Toy     3.50    6
2       Sock    7.50    9

1. Add
2. Delete
3. Update
4. View Products
5. Exit
3
Enter product id, new price, new quantity
4 5 10
Id not found
```

Note that as an admin we can update existing quantities. If an invalid id is supplied, an appropriate error message is given as shown above. The admin's work is done for now. We can exit the program and take a look at the LOG file generated. Notice how the LOG file only keeps track of successful transactions. (It doesn't store the invalid update above).

log.txt

```
1   ADD Box 10.50   12
2   ADD Toy 2.50    5
3   ADD Sock    7.50    9
4   UPDATE  1   3.50    6
5
```

Next we shall connect to the server as a user.

```
[adsultimate@fedora Shopping]$ ./a.out
Welcome to online shopping! Connecting to server...
Success!

1. Admin
2. User
2

1. View Products
2. View Cart
3. Add to Cart
4. Checkout
1
PRODUCTS:
0       Box     10.50   12
1       Toy     3.50    6
2       Sock    7.50    9

1. View Products
2. View Cart
3. Add to Cart
4. Checkout
3
Enter product id and quantity
1 2
Added to cart

1. View Products
2. View Cart
3. Add to Cart
4. Checkout
3
Enter product id and quantity
2 3
Added to cart

1. View Products
2. View Cart
3. Add to Cart
4. Checkout
2
CART:
1       Toy     3.50    2
2       Sock    7.50    3
```

In the above image, we have added two items into our cart. A toy and a sock. Next we can proceed to checkout but first we are given the option to edit our cart.

```
1. View Products
2. View Cart
3. Add to Cart
4. Checkout
4

Edit your cart before proceeding!

1. View Cart
2. Update
3. Delete
4. Proceed to purchase
2
Enter productID and new quantity
1 3
Successfully updated quantity

1. View Cart
2. Update
3. Delete
4. Proceed to purchase
1
CART:
1       Toy     3.50    3
2       Sock    7.50    3
```

After finalizing our cart, we can proceed to payment. Now the items get locked, meaning they cannot be accessed by other processes until the payment is confirmed. The latest prices and quantity are accessed from the server and if sufficient quantity is available, success is returned. And the total price is shown.

```
Entering payment gateway....

Final Bill: (Note that items which exceed the quantity have been removed)
1       Toy     3.50    3       SUCCESS
2       Sock    7.50    3       SUCCESS
Total: 33.00
Enter (y) to confirm
```

To test out the locking mechanism, we will open another user simultaneously. Notice how we can still view the available products as that does not require locking.

```
[adsultimate@fedora Shopping]$ ./a.out
Welcome to online shopping! Connecting to server...
Success!

1. Admin
2. User
2

1. View Products
2. View Cart
3. Add to Cart
4. Checkout
1
PRODUCTS:
0       Box     10.50   12
1       Toy     3.50    6
2       Sock    7.50    9

1. View Products
2. View Cart
3. Add to Cart
4. Checkout
3
Enter product id and quantity
1 1
Added to cart

1. View Products
2. View Cart
3. Add to Cart
4. Checkout
2
CART:
1       Toy     3.50    1
```

Here we are purchasing a common item, toy with id 1. Therefore the second process is still waiting for

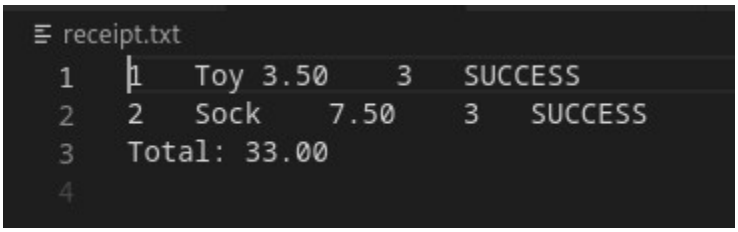
```
Edit your cart before proceeding!

1. View Cart
2. Update
3. Delete
4. Proceed to purchase
4

Entering payment gateway....
```

the first to finish at the payment gateway.

After confirming the first payment by entering (y), our receipt is shown. And then the other process is free to continue.




```
4

Entering payment gateway....

Final Bill: (Note that items which exceed the quantity have been removed)
1       Toy     3.50    1       SUCCESS
Total: 3.50
Enter (y) to confirm
y

Thank you for shopping. Your receipt has been created
```

If we look at the server program we can see the clients and the requests they have made.

```
[adsultimate@fedora Shopping]$ gcc server.c
[adsultimate@fedora Shopping]$ ./a.out
Waiting for connections
Client joined
Choice 1 selected
Choice 1 selected
Choice 1 selected
Choice 4 selected
Choice 3 selected
Choice 4 selected
Choice 3 selected
Client disconnected
Client joined
Choice 4 selected
Choice 6 selected
Choice 6 selected
Choice 7 selected
Client joined
Choice 4 selected
Choice 6 selected
Choice 7 selected
Client disconnected
Client disconnected
```

Now let us open some more users.

```
[adsultimate@fedora Shopping]$ ./a.out
Welcome to online shopping! Connecting to server...
Success!

1. Admin
2. User
2

1. View Products
2. View Cart
3. Add to Cart
4. Checkout
1
PRODUCTS:
0       Box     10.50   12
1       Toy     3.50    2
2       Sock    7.50    6

1. View Products
2. View Cart
3. Add to Cart
4. Checkout
3
Enter product id and quantity
0 5
Added to cart

1. View Products
2. View Cart
3. Add to Cart
4. Checkout
3
Enter product id and quantity
1 5
Added to cart
Please note the requested quantity is not currently available
```

Note that the previous purchases have reflected in server now as the quantities have decreased. This time we will purchase more quantity than that which is available. The program gives us a warning but doesn't prevent us from proceeding as until the section becomes locked, the quantities may change.

```
Entering payment gateway....

Final Bill: (Note that items which exceed the quantity have been removed)
0       Box     10.50   5       SUCCESS
1       Toy     3.50    5       NOT AVAILABLE
Total: 52.50
Enter (y) to confirm
```

But once the item is locked, the program is sure that the requested quantity is unavailable. So it becomes excluded from the total bill. Now let's open another process and try purchasing an item which is not locked, such as the sock.

Notice how we are able to purchase the sock concurrently as it is not locked by the previous process. If we enter (n) instead of (y), the quantities won't be removed from the stock and the receipt will not generate.

Mechanisms Used: In this section, the working of the code will be explained. Firstly, the server works by creating a new thread every time a client connection is accepted. This thread is in charge of handling only this client, and only exits after client disconnects. The client can make several type of requests to the server either as admin or user, defined as a variable choice. There are a total of 7 choices, and these choices are displayed in the server as shown in the above picture.

Choice 1 - Admin add
Choice 2 – Admin delete
Choice 3 – Admin update
Choice 4 – View Products
Choice 5 – *Not in use anymore*
Choice 6 – Add product to cart
Choice 7 – Make purchase

Depending on the choice, the server sends different data back to the client. So any data sent through a socket from client to server is preceded by the choice value.

The product data is stored in server in a global array of structs which can be accessed by all the threads. This array has a max size of 1024. To protect this array from race conditions, semaphores are used. We create 1024 binary semaphores, one for each index of the array. Whenever a critical section is entered, the semaphore is used. Given below are the critical sections of the program:

1. Any kind of admin operation which adds, deletes, or modifies a products
2. When the user enters the purchase gateway

In the second case, the server iterates through the items in the user's cart and locks all of them before modifying them. Because all items are locked beforehand, deadlocks are prevented.

Finally a log file is also used to ensure data is preserved in case of corruption. So any admin update causes the thread to write into the log file, the detail of the operation. One extra semaphore is also used here to prevent race conditions while writing into the file.