Yukta Arvind Rajapur
IMT2021066

OS Mini Project

Online Retail Store

OVERVIEW

This is a server-client-based project where the server is Admin and the user is the client. Their communication between the user and admin happens using socket programming. Here the customer is the client that can shop for products in a store. Admin is similar to the store owner who controls the stock and price of each product. When the admin updates some product, or a customer has some products in their cart, other users cannot access these. This is implemented using file locking.

FILES

1. Client.c - The client-side code where the customer interacts with the store by sending requests
2. Server.c - The server-side code that handles the requests from the client, implements locking.
3. Products.txt - Text file with all the products in the store, each record stored as a struct product
4. Orders.txt - text file with all the cart items or respective customers
5. Index.txt - An index file used to map a customer id to a cart id.
6. Receipt.txt - File containing the total bill after a customer is done shopping
7. AdminLog.txt - Files stores summary of updates made by admin in the store.

HOW TO RUN

Open 2 terminals side by side, one for the client and one for the server. Both of the programs will run concurrently.

Server.c -

$ gcc -o server server.c

$./server

client.c -

$ gcc -o client client.c

$./client

In the client terminal, a prompt will appear asking you to log in as an admin/customer. On choosing a desired option, an option-based menu will appear through which you can interact with the store.

***Login as a customer** - You will first have to generate a unique customer id that will be linked to your cart. This allows multiple customers to shop at once. Press 0 to generate an id. For every method a prompt will appear to enter your id, hence generate one before proceeding.*

OS Concepts Used

- **Socket Programming** (explained in detail further)
- **File Locking** (explained in detail further)
- **File handling** - File descriptors, **lseek(), read(), write()** system calls are used to read and write to files

Product_cart.h

Header file containing the definition of structures used in the store.

-struct product, struct cart, struct index

- The index here is used to store a key/offset value pair in an index file that contains the map from customer-id to cart_id. It is needed when we want to fins the cart associated with a customer or assign a new customer id.
- The maximum number of items in a single cart is 50, it is defined by a global.

```
#define MAX_CART 50

struct index{
    int key;
    int offset;
};

struct product{
    int prod_id;
    char pname[100];
    int cost;
    int qty;
};

struct cart{
    int cust_id;
    struct product items[MAX_CART];
};
```

Client.c

Methods available for a customer

On first executing the code, if the user is a new customer they have to register first. This will give them a unique customer id that will act as a key to your cart and any methods you want to use.

The following option-based menu pops up-

```
What would you like to do? If you're a new customer please register first. Enter 0 to register.
1)View all items
2)View your cart
3)Buy a product
4)Edit cart
5)Confirm cart and go to payment
6)Exit
```

By entering any of the given numbers, the respective method will execute.

1. View all items

```
Connecting to store...
ID      Name    Stock   Cost
2       apple   11      25
3       banana  20      10
4       tea     13      400
```

2. View your cart

```
Customer ID 1
ID      Name    Quantity        Price
1       mango   10      50
1       mango   3       50
2       apple   4       30
```

3. Buy a product

```
Enter choice:3
Enter customer id:
1
Enter product id:
2
Enter product quantity:
4
Item added to cart
```

4. Edit cart

```
Enter customer id:
1
Enter product id:
1
Enter product quantity:
10
Update successful
```

5. Confirm cart and go to payment

```
Enter customer id:
3
Here is a summary of your cart
Product id- 2
Ordered - 1; In stock - 1; Price - 25
---end----
Total amount to be payed: 25
```

```
Total amount to be payed: 25
Enter amount you want to pay: 25
-------Payment successful!------
```

Implementation and Design Decisions

Since some methods use repeated pieces of code, I decided to use functional programming to break down the code. It makes it much more efficient and cleaner to understand. In client.c the following functions are present:

- displayproduct() - Fetches details of product struct and prints in the terminal.
- displaystore() - Reads the file of all store items and prints each product using the above function
- getTotal() - Takes unique cart id and calculates the total price of products in the cart from the cart file.
- generateReceipt() - Prints summary of bill in the terminal and generates receipt text file for checkout and final payment
- displayMycart() - Displays products in a unique customer's cart

All these are self-explanatory by their name.

Connection to the server using Socket programming

The client-side code creates a socket file descriptor as one endpoint channel to communicate with the server. The main functions used are **socket()** and **connect()**. Below is the actual code implemented.

```
struct sockaddr_in client;
int socketfd, new_sd, connectfd;

socketfd = socket (AF_INET, SOCK_STREAM, 0);//creating socket filed
if(socketfd == -1)
{
    perror("socket error: ");
    return -1;
}

client.sin_family = AF_INET;
client.sin_addr.s_addr = INADDR_ANY;
client.sin_port = htons (PORT);

connectfd = connect(socketfd, (struct sockaddr_*)&client, sizeof(client)); //connecting to server
if(connectfd == -1)
{
    perror("connect failed");
    return -1;
}
```

Methods available for Admin

There is a single admin user unlike a customer, there cannot be multiple. The admin control the overall data in the store. Data is stored in various files as mentioned earlier and this is synchronized using file locking. The following option-based menu shows up containing all the methods for admin.

```
----------ADMIN----------
What you want to do
1)Add a product
2)Modify price of existing product
3)Modify quantity of existing product
4)Delete product
5)View all products in store
6)Exit
Enter:
```

1. Add product

```
Enter:1
Enter product id:
2
Enter product name:
apple
Enter product price:
31
Enter product quantity:
55
Added successfully
```

2. Modify price/quantity of existing product

```
Enter:2
Enter product id:
2
Enter product price:
30
Price modified
```

3. Delete product

```
Enter:4
Enter product id:
3
Delete successful
```

4. View all products

```
Connecting to store...
ID      Name    Stock   Cost
1       mango   100     50
2       apple   55      30
3       grapes  10      10
```

Server.c

The server handles the incoming requests from all the users. The main communication channel is the socket fd. **read()** and **write()** calls are used to send and receive data from the client and server.

Connection to the client using Socket programming

The **socket()** system call along with the series of functions-**bind(), listen(), and accept()** are the main system calls used. The following shows how this was implemented.

```
int socketfd = socket (AF_INET, SOCK_STREAM, 0);
if(socketfd == -1)
{
    perror("socket");
    return -1;
}

struct sockaddr_in server, client;
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(PORT);

if(bind(socketfd, (struct sockaddr *)&server, sizeof(server)) == -1)
{
    perror("bind");
    return -1;
}
if(listen(socketfd, 5) == -1)
{
    perror("listen");
    return -1;
} //wait for client

printf("server ready\n");

while(1)
{
    int newsd = accept(socketfd, (struct sockaddr *)&client, sizeof(client));
    if(newsd == -1)
    {
        perror("error newsd");
        return -1;
    }

    if(!fork())
    {
        printf("Connection established with client");
```

Locking

File locking- uses these methods to lock and unlock the respective files as per the function name. The main system calls used here were **struct flock and fnctl():** The following functions are used to implement read and write locks on files whenever an admin is changing products or the user is going to buy the items in the cart.

- ReadProductLock()-read lock on products.txt
- WriteProductLock()-write lock on products.txt
- CustomerLock()-In the index.txt file which contains key as customer id and cart id, while adding new customers this is locked
- Unlock()- A general unlock function used by all lock functions when they need to unlock.
- LockCart() - this takes a parameter option (1 or 2) 1 indicates read lock and 2 indicates write lock.

Implementation and Design Decisions

Since some methods use repeated pieces of code, I decided to use functional programming to break down the code. It makes it much more efficient and cleaner to understand. The following functions are present for admin:

Admin side functions -

- showProducts() - Prints products already present in the store
- updateProduct() - Allows admin to modify the price or quantity of the product
- deleteProduct() - Allows admin to delete the product. This is implemented by making the product id = -1, which is invalid
- generateAdminLog() - Writes the changes made by admin in a text file

Client-side functions -

- AddCustomer() - generates new customer id and writes it into the index file
- ViewCart() - Displays items in the cart of a customer
- BuyProduct() - Adds new product to cart of customer by writing into cart file
- EditCart() - Allows customer to change the quantity of a particular item in the cart
- GetPayment() - Proceeds to checkout and locks all the items in the cart.

All these functions in the server code help the corresponding requests that come in from the client. For eg, if a customer wants to buy a product, BuyProduct() function adds that product to the cart and sends a confirmation response back to the client. Similarly for admin functions. They implement file locking internally using the functions above while modifying files.