OS - Mini Project

Online Retail Store

Brij Desai (IMT2021067)

Overview:

This project is designed to replicate an OnlineStore through a terminal-based interface, using a client-server architecture. Client requests are transmitted to the server through sockets, which are then processed and returned with the relevant results. User privileges are determined based on their type, with customers having access to their own carts, while Admin users are granted additional privileges to modify store items. The communication between the client and server is facilitated through Socket programming, while system calls are used for interaction between the client and server. Data files are accessed by the server through file locking mechanisms.

File Structure:

- Client.c: The client-side code where the customer interacts with the store by sending requests
- Server.c: The server-side code that handles the requests from the client, implements locking.
- Structs.h: File containing the structs and macros to be used in the program.
- inventory.txt: Text file with all the products in the store, each record stored as a struct product.
- carts.txt: text file with all the cart items or respective customers.
- customers.txt: this file contains the customers registered with the store.
- adminLog.txt: this file is used to store the history of the updates, deletes and additions of the products being done by the admin.
- receipt.txt: File containing the total bill after a customer is done shopping.

Steps to Run:

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

Server.c -
$ gcc -o server Server.c
$./server

client.c -
$ gcc -o client Client.c
$./client

Structs.h

In order to exchange data between the server and client, the program utilizes a set of macros and structs that are consistently used throughout.

- Struct product: the struct to store the product info (product id, product name, quantity and price).
- Struct cart: the struct to store the cart info for a customer (customer id, list of products in the cart).
- Struct cart_index: struct which contains the customer id and the cart offset.

Client.c

Methods available for customer

The program then takes the input for the type of user (1 for customer, 2 for admin). During one run of the program, the user can only be a customer or an admin, he can't switch.
If user is chosen, the customer menu is displayed and customer functions are performed. The choice of the user is sent to the server to notify the server about the same.

- The user can register as a new customer. This is implemented by calling the registerCustomer() function on the server side.
- In order to display the available products, we retrieve the inventory information from the server by calling the printInventory() function. On the server side, the listInventory() function is called.
- To obtain a customer's cart, we prompt the user to provide their customer id, and subsequently retrieve the cart from the server. If the customer id is incorrect, we display an appropriate error message. On the server side, the viewCart() function is called.
- To add a product, we ask the user for the customer id, product id, and quantity. If any errors are encountered during this process, we notify the user accordingly. Otherwise, the server's cart is updated with the selected product. On the server side, the addProductToCart() function is called.

- The procedure for editing a product is the same as for adding a product. On the server side, the editCartItem() function is called.
- To process payment for the items in the cart, the cart is first presented to the user for review, after which the total cost is calculated and displayed. The user is then prompted to input the amount they wish to pay. If the correct amount is entered, a receipt is generated. Otherwise, the user is prompted to enter the amount again. On the server side, the payment() function is called.
- To exit the menu, we can simply break the loop.

Methods available for Admin

If the user selects the admin option, the admin menu is displayed and the corresponding admin functions can be performed.

- In order to add a new product, the user provides the product ID, product name, current stock quantity, and price. This information is then relayed to the server, which adds the new product to the inventory. If any errors occur during this process, they are reported to the user. On the server side, the addProducts() function is called.
- To delete a product from the inventory, the user simply enters the product ID. If the ID is correct, the corresponding product is deleted. If not, an error message is displayed. When a product is deleted by an admin, it is not immediately reflected in the carts of customers who have already added the product. However, once a customer proceeds to the payment page, they will see the updated availability of the product. On the server side, the deleteProduct() function is called.
- To modify a product's price or quantity, the user enters the product ID along with the updated price or quantity. If no errors occur, the product information is updated and the user is notified accordingly. When a price or quantity is updated by an admin, the old values may still be visible in customers' carts. However, once the customer proceeds to the payment page, they will see the updated information. On the server side, the updateProduct() function is called.
- To display the available products, we obtain the inventory information from the server by calling the printInventory() function, and then display it to the user. On the server side, the listInventory() function is called.
- To exit the admin menu, we simply break out of the loop.

Server.c

- To prevent race conditions amongst multiple clients, the application uses extensive file locking that is implemented using a number of functions which are called whenever we are required to access the data files. These functions are: **inventoryReadLock(), inventoryWriteLock(), cartRecordLock(), readLockCust() and unlock()**.
- **getOffset()**

The server code calls this function internally in several places. It searches for the cart offset of a given customer id by iterating over the customers.txt file. If the offset is found, it is returned. Otherwise, it returns -1. The cart offset is used to update a user's cart whenever necessary.

- **registerCustomer()**
  By calculating the maximum of previous customer IDs and adding 1, this function generates a new customer ID and adds a new customer to the system.
- **listInventory()**
  This function displays the products available in the inventory and is used by both customers and admin.
- **addProducts()**
  The purpose of this function is to enable the administrator to add a new product to the inventory. It receives the necessary information about the product and checks whether the product ID already exists. If it does, an error message is displayed; otherwise, the product is added to the inventory. A log entry is also created in the adminLog.txt file to record the addition.
- **updateProduct()**
  This function is utilized to modify the price or quantity of a product in the inventory. If the quantity parameter is passed as 0, then the delete product function is called. Otherwise, the appropriate update is performed based on the ch argument passed into the function. When opt = 1, the quantity is updated, otherwise the price is updated. A log statement is written to the adminLog.txt file to record the modification.
- **deleteProduct()**
  This function is employed by the admin to remove a product from the inventory. Initially, it verifies whether the product id is valid or not. If it is valid, the product is deleted. Otherwise, an appropriate error message is displayed. A log statement indicating the action is written to the adminReceipt.txt file.
- **viewCart()**
  The function takes a customer id as an input and uses the getOffset() function to obtain the cart offset. If the offset is invalid, i.e., -1, an appropriate message is displayed to the client. Otherwise, the correct cart is retrieved and sent to the client.
- **addProductToCart()**
  This function is designed for adding a product to a cart. It receives the customer ID, product ID, and quantity as inputs from the client. The function then verifies the validity of the customer ID and checks if the product already exists in the cart. It also confirms whether the requested quantity is available in stock. If all the aforementioned conditions are met, the product is added to the customer's cart. However, the function also checks the PROD_LIMIT macro for the number of products that can be in a cart before adding. If the limit is exceeded, an error message is displayed.
- **editCartItem()**
  The function allows users to modify the quantity of items they have already ordered. The client provides the customer id, product id, and new quantity. The function performs the same checks for customer id and new quantity as the addProductToCart() function. It also verifies if the product id is already in the cart. The remaining steps of the function

are similar to those in addProductToCart(). If the input is valid, the product is updated in the user's cart; otherwise, an error message is displayed.

- **generateAdminLog()**
  When the admin completes his updates and exits the program, this function is called to indicate the same. The function writes the updated inventory to the adminLog.txt file, reflecting all the changes made by the admin.
- **payment()**
  The original inventory is not updated when there are changes in quantity because the customer has not yet purchased the products. The changes are only reflected in the inventory when the customer buys the products. Before the customer pays for their cart, the system checks whether the originally requested quantity is still available, as it may have been sold to another customer in the meantime. The total amount due is updated based on the current stock availability. To complete the payment process, the customer enters the amount displayed on the screen. Once the payment is made, all the changes are recorded in the inventory, the customer's cart is emptied, and the purchased items are added to the receipt.txt file as a receipt.

OS concepts used

Socket Programming

The communication between the server and the client in the program is facilitated by sockets. At the server side, socket system calls such as socket, bind, listen, and accept are used to establish the socket, while at the client side, socket system calls such as socket and connect are used.
A concurrent server is set up using the fork() system call.
All reads and writes involving the socket employ system calls such as read and write.

(Server side socket setup)

```
int sd = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in serv, cli;

serv.sin_family = AF_INET;
serv.sin_addr.s_addr = INADDR_ANY;
serv.sin_port = htons(6767);

int opt = 1;
if (setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)))
{
    perror("Error: ");
    return -1;
}

if (bind(sd, (struct sockaddr *)&serv, sizeof(serv)) == -1)
{
    perror("Error: ");
    return -1;
}

if (listen(sd, 5) == -1)
{
    perror("Error: ");
    return -1;
}
```

(Setting up concurrent server)

```
while (1)
{
    int nsd = accept(sd, (struct sockaddr *)&cli, &size);
    if (nsd == -1)
    {
        return -1;
    }

    if (!fork())
    {
        printf("Connection with client successful\n");
        close(sd);

        int user;
        read(nsd, &user, sizeof(int));
        char ch;

        if (user == 1)
        {
            while (1)
            {
                read(nsd, &ch, sizeof(char));
```

(Client side socket setup)

```
printf("Connecting to server...\n");
int sd = socket(AF_INET, SOCK_STREAM, 0);

if (sd == -1)
{
    perror("Error: ");
    return -1;
}

struct sockaddr_in serv;

serv.sin_family = AF_INET;
serv.sin_addr.s_addr = INADDR_ANY;
serv.sin_port = htons(6767);

if (connect(sd, (struct sockaddr *)&serv, sizeof(serv)) == -1)
{
    perror("Error: ");
    return -1;
}
```

File locking

The program utilizes both mandatory and record locking in various parts of the code:
A mandatory read lock is imposed on the entire customers.txt file when searching for a specific customer ID.
Once the offset of a customer's cart in carts.txt is found, record locking is applied on that specific cart to ensure it is locked for either reading or writing.
Mandatory read locking is applied to all products when attempting to find a specific product ID, as well as when displaying the inventory.
To modify a specific product, the read lock on the products is removed and replaced with a write lock to enable the necessary changes to be made.