Mini Project - OnlineStore

Siddharth Kothari (siddharth.kothari@iiitb.ac.in)

1 Overview

This is a terminal based project to simulate the working of an OnlineStore. The project uses a client server architecture. Requests from the client are sent using sockets to the server program which processes the request, and returns the result needed. Based on the type of user, there are privileges associated with them. A normal user is a customer who can see his own cart, while the Admin user has special privileges to update the items in the store.

The program uses Socket programming for communication between the server and the client. The client side and server side interact using system calls, and the server side interacts with the data files using file locking mechanisms.

2 Application Architecture

The application has the following files:

- Client.c - this file contains the code to be run at the client side.
- Server.c - this file contains the code to be run at the server side.
- customers.txt - this file contains the customers registered with the store.
- orders.txt - this file contains the carts for each of the users.
- records.txt - this file contains the inventory of the store, i.e. the products currently in the store with their quantities and prices.
- receipt.txt - this file contains the receipt for the user after payment.
- headers.h - this file contains the structs and macros to be used in the program.

3 How to Run the Code?

1. Open 2 terminal windows.
2. In the first terminal window, run the commands

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

   This starts up the server.
3. In the second terminal window, run the commands

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

   This connects the server to the client.
4. We can then run the program as is directed by the client program.

4 headers.h

The following structs and macros are used throughout the program to pass data among the server and client.

- 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, the list of products in the cart).
- MAX_PROD - the maximum products in a cart. This is set to 20.
- struct index - the struct which contains the customer id and his cart offset in the orders.txt file (thus customers.txt serves as an ndx file to orders.txt).

5 Functionalities of the user

Normal User (Customer) The program intends to provide the following functionalities to customers (normal users):

- To register himself as a new customer
- To view the items available at the store with their respective quantities in stock and the prices, so that the customer can know what the store offers
- To add product to his cart, along with the quantity of the product to be ordered
- To edit the quantities of existing products in his cart, this also allows him to delete items from his cart.
- To proceed for payment for all the products in his cart. Following successful payment, a receipt is generated for the user to view his order.

Admin User The admin user serves as the administrator of the OnlineStore. He can manage the products in the online store, and has the following functionalities available to him:

- To add a new product to the store
- To update the quantity available or the price of an existing product in his store.
- To explicitly remove products from the store.
- To see his inventory, i.e. the items currently present in the store.

6 Functionalities of the server

Server The server serves the requests of the client (whether a customer or the admin) and has functionality to support the above requests.

7 Implementation of Client.c

7.1 Helper functions

The code uses the following helper functions for modularisation:

1. displayMenuUser(), displayMenuAdmin()
   These functions are called inside while loops to display the options available to the customer / admin. Looking at these menus, the user enters his choices. (Customer has option letters a-g, admin has a-f).
2. printProduct(), getInventory()
   These functions are used to display a formatted report to the user, after the client gets a response from the server.

3. calculateTotal(), generateReceipt()
   These functions are used to calculate total in the user cart, and generate receipt once the payments are done. These functions are called when the user enters the option to pay for the items present in the cart.

4. custIdTaker(), productIdTaker(), quantityTaker(), priceTaker()
   These functions are used to take input from the user and enforce the conditions that all these values are non negative. Negative values for these quantities may give unpredictable results, and are not desirable or needed. These functions enforce the constraints on the inputs.

7.2 Socket Setup

The main function first consists of socket setup, code for which is as follows:

```
int main(){
    printf("Establishing connection to server\n");
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);

    if (sockfd == -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(5555);

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

7.3 Implementation of Customer functions

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 customer functions are as described above, and the implementations are listed below (all inputs taken from the user use the above defined user input functions):

1. To exit the menu, we simply break the loop.

2. To display products, we use the getInventory() function to get a response from the server, and display it.

3. To get the cart of a customer, we take the customer id as input, and return the cart given by the server, or return a message if the customer id is wrong.

4. To add a product, we take customer id, product id and quantity as inputs. In case of errors, we communicate the error to the user, else the cart gets updated at the server side.

5. To edit a product, the procedure is the same.

6. To pay for the cart, the cart is first displayed to the user, following which the total is calculated and displayed to the user. The user is then asked to enter the amount to pay. If correct amount is entered, we generate the receipt, else we ask the user to enter it again.

7. Finally to add himself as a new customer, the customer just enters a confirmation, and the auto generated customer id is given to the user.

7.4 Implementation of Admin Functions

If admin is chosen, the admin menu is displayed and admin functions are performed.
The admin functions are as described above, and the implementations are listed below (as stated previously, all inputs are taken using the helper functions):

1. To add a product, we take the product id, product name, quantity in stock and the price as inputs from the user, and communicate the same to the server, which adds the product to the inventory, or reports the error.

2. To delete a product from the inventory, we simply ask for the product id. If the product id is correct, the product id is deleted, else error message is displayed. Products deleted by the admin are not immediately updated in the cart, and even after deletion, products are visible. However once the customer goes to his payment page, he once again sees the updated values. Price and quantity = 0 indicate that the product is no longer available.

3. To edit a product's quantity / price, we take the product id of the product, and the new quantity / price. If no errors, the product is edited and the appropriate message is displayed, else appropriate error message is returned. Prices and quantities updated by the admin are not immediately seen in the cart, and old values are visible if the user sees his cart. However once the customer goes to his payment page, he once again sees the updated values.

4. To display products, we use the getInventory() function to get a response from the server, and display it.

5. To exit the menu, we simply break the loop.

8 Implementation of Client.c

8.1 Helper functions

The code uses the following helper functions for modularisation:

1. setLockCust(), unlock(), productReadLock(), productWriteLock(), CartOffsetLock()
   These functions are file locking functions. They are called whenever we are required to access the data files. These functions lock the corresponding portions of the file and we then access that portion in the function. The descriptions are as follows:

   - setLockCust() : This sets a mandatory read lock on the entire customers.txt. This is used when we want to see the current registered customers with the store.
   - productReadLock() : This sets a mandatory read lock on the entire records.txt file. This is used when we want to check particular products or display them.
   - productWriteLock():This sets a write lock on the current product, in order to update it. This is used when we want to update or delete a product.
   - CartOffsetLock(): This takes the cartOffset as an argument (which we get from the getOffset() function) and locks that particular cart for reading/writing.
   - unlock(): to unlock any given lock.

   From now on, we assume that whenever a file access is done, the appropriate lock is implemented, and unlocking is also done wherever needed.

2. getOffset()
   This function is internally called by many of the functions in the server code. This code iterates over the customers.txt file and finds the cart offset for the given customer id. If it is found, return the offset, otherwise return -1. This offset is then used whenever we need to update the cart of a user.

3. addProducts()
   This function is used by the admin to add a product to the inventory. It takes the product parameters, and checks if the product id is duplicate or not. If it is, the appropriate message is printed, else the product is added.

4. listProducts()
   This function is used to display the products from the inventory. Can be used by both admin and user.

5. deleteProduct()
   This function is used by admin to delete a product. It first finds whether the product id is valid or not. If it is, it deletes the product. Else it prints appropriate error message.

6. updateProduct()
   This is used to update the price or quantity of the product in the inventory. If 0 is passed as the quantity, then delete product is called. Else the corresponding update is performed. This is decided by the ch argument passed into the function. When ch = 1, we update price, else the quantity.

7. addCustomer()
   This function adds a new customer with an auto generated customer id. The customer id is generated by calculating the max of previous customer ids + 1.

8. viewCart()
   This takes the customer id as an argument, and calls the getOffset() function to get the cart offset. The cart offset is -1 if it is invalid, else we get the correct cart which we write to the client.

9. addProductToCart()
   This function is used to add a product to a cart. The function reads the customer id, the product id and the quantity from the client. If the customer id is invalid or the product id already exists in the cart, the corresponding message is printed. It also checks whether the quantity requested is in stock or not. Only if all these conditions are satisfied, then we add the product to the customers' cart. There is also a limit of MAX_PROD on the number of products that can be in a cart. This condition is also checked before adding, else error message is printed.

10. editProductInCart()
    The user can only change the quantity of an item he has already ordered. The function reads the customer id, product id and new quantity from the client. It performs the same checks for customer id and new quantity as in addProductToCart(). The product id is also checked whether it is currently in the cart or not. The rest of the functionality resembles the addProductToCart(). If everything is correct, the product is updated in the user's cart, or else an error message is displayed.

11. payment()
    Any changes in quantity are not reflected in the original inventory, as the customer has currently not bought the products. Only when he buys the products are the changes reflected in the inventory. When the customer wants to pay for his cart, we first see whether the quantity originally requested was in the cart or not (as some other customer may have bought it in the meanwhile). The total is thus reflected after taking the current stock in consideration. For the payment, the user just has to enter the amount displayed on screen. Once this is done, all changes are recorded in the inventory, the cart of the customer is cleared off, and the items that the customer just bought are reflected in the receipt.txt file as a receipt.

8.2 Socket Setup

The main function first consists of socket setup, code for which is as follows:

```
int main(){
    printf("Setting up server\n");
    int fd = open("records.txt", O_RDWR | O_CREAT, 0777);
    int fd_cart = open("orders.txt", O_RDWR | O_CREAT, 0777);
    int fd_custs = open("customers.txt", O_RDWR | O_CREAT, 0777);
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1){
        perror("Error: ");
        return -1;
    }
    struct sockaddr_in serv, cli;
```

```
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = INADDR_ANY;
serv.sin_port = htons(5555);
int opt = 1;
if(setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))){
    perror("Error: ");
    return -1;
}
if (bind(sockfd, (struct sockaddr *)&serv, sizeof(serv)) == -1){
    perror("Error: ");
    return -1;
}
if (listen(sockfd, 5) == -1){
    perror("Error: ");
    return -1;
}
int size = sizeof(cli);
printf("Server set up successfully\n");
```

Following this, any client requests are accepted, and a fork() is called to implement a concurrent server mechanism.

8.3 Implementation of Server functions

The program reads the input sent by the clients about the type of user and the choice of the user, and calls the appropriate helper functions for the case.

User functions -

1. If the user asks to exit the program, we break the loop.
2. If the user asks to see the inventory, we then call the listProducts() function to list the products.
3. If the user asks to view his cart, the viewCart function is called().
4. If the user asks to add a product to his cart, the addProductToCart() function is called.
5. If the user asks to edit a product in his cart, the editProductToCart() function is called.
6. If the user asks to pay for his cart, the payment() function is called.
7. Finally to add the user as a new customer, call the addCustomer() function.

Admin functions

1. If the admin asks to add a product to the store, the addProducts() function is called.
2. If the admin asks to delete a product from the inventory, the deleteProduct() function is called.
3. If the admin asks to edit the price of a product in the inventory, the updateProduct() function is called, with ch set to 1.
4. If the admin asks to edit the quantity of a product in the inventory, the updateProduct() function is called, with ch set to 2.
5. If the admin asks to see the inventory, we then call the listProducts() function to list the products.
6. If the admin asks to exit the program, we break the loop.

Thank you