OS PROJECT REPORT

In this project, I have implemented a basic client-server architecture for buying and selling of products. The following project uses the following concepts from operating system: -

1) File management
2) File locking
3) Locking using semaphores
4) Socket programming

Code Overview: -

This is a server-client based project where the server is Admin and the user is the client. There communication between the user and admin should happen using socket programming.

Whenever a client is accessing the program, it has to either signup if it's a first time user or login using appropriate username and password.

The functionalities of **Admin** are as follows: -

1)Authentication: -

```
struct Admin{
    char name[MAX_SIZE];
    char password[MAX_SIZE];
};
```

This structure stores the username and password for authentication, and all the authentication data is stored in admin_details file.

2)Adding/Deleting a product: -

```
struct product
{
    int productid;
    char product[MAX_SIZE];
    float price;
    int quantity;
};
```

Addition and deletion of products take place in product_list file. Whenever a delete is occurring, the file is locked using file locking by using fcntl() method. Appropriate messages are displayed if the client asks for deleting a product which does not exist in the first place.

3)Updating the price/quantity: -

Updation to product list by price/quantity is only possible if the following conditions are met: -

1) Product Id given is correct
2) The new quantity must be greater than or equal to 0. If the updated quantity is zero, the product is removed from product_list file.
3) The new price should be greater than 0.0

If any of the following conditions are not met, updation is failed.

4)Logout : -

Logout feature is also given in case the work of admin is over. After logout, all the products and their details are shown as a log.

The functionalities of **User** are as follows: -

1) **Authentication: -**
   The user has the option of either signing up or logging in based on their choice. All the authentication details are stored in user_auth_details. In case of user logging, each session generates a unique user id which is used for their edit and adding into the cart, and generating log files during payment.

2) **Displaying all products: -**
   It displays all the products in the format: P_ID, P_Name, Cost using product_list file.

3) **Displaying the cart : -**
   It displays the cart items added by the user. All the cart details are stored in cart_details file and corresponding to each cart data, a user id is stored so as to identify the user to which the cart belongs.
   The structure of Cart Items look like: -

```
struct CartItems{
    int user_id;
    int product_id;
    char product_name[MAX_SIZE];
    int quantity;
};
```

4) **Adding items to the cart**

When choosing this option, user just needs to enter the product name and the quantity he/she wants, and then the item is added to the cart. The quantity entered must be greater than 0, otherwise adding to the cart will be failed.

5) Editing the cart

We can make amends to the quantity of the products. If we want to delete an item from the cart, just enter it's quantity as 0 and it will be deleted.

6) Making payment

This feature makes use of semaphores for locking the file, and after the payments have been done, the quantity is updated in the product list. We make use of this structure to generate the receipt: -

```
struct log
{
    int user_id;
    char productid;
    float price;
    int quantity;
};
```

Now during the payment, two cases might arise: -

1) Quantity is less than available quantity: -

```
if (cart.quantity <= prod.quantity) {
    reduceQuantity(socket_descriptor, cart.product_id, cart.quantity);
    generateLog(cart.product_id, cart.quantity, prod.price);
    total_price += cart.quantity * prod.price;
```

We simply reduce the quantity and add the receipt item to the log.

2) Quantity is more than available quantity: -

```
printf("We have only %d quantity of %s.\nDo you want to buy (1-Yes,
2-No)\n", prod.quantity,
        prod.product);
int yes_no;
scanf("%d", &yes_no);
if (yes_no == 1) {
    reduceQuantity(socket_descriptor, cart.product_id, prod.quantity);
    generateLog(cart.product_id, prod.quantity, prod.price);
    total_price += prod.quantity * prod.price;
```

If the user wants to buy the available quantity, he chooses 1 and then the product is removed from product_list file and receipt is added in the log.

At the end of the payment, the user is given the receipt and the semaphore lock is removed from the file.

7) Logout

We exit from the client whenever the user wishes to end their purchase.

I have ensured that since server is the admin, therefore all the functions of the admin will be handled by server and displayed to the client side by the help of socket descriptors, and since user is the client, all the client processes have to be managed in the client side of the code and server does not access user functionalities.
