OS Mini-Project

Vikas Kalyanapuram - IMT2021040

May 11, 2023

Github Repo of Project - https://github.com/LieutPaul/OS-Mini-Project

1 Problem Statement

This is a terminal based project to simulate the working of an e-commerce website. It works on a client-server model where the communication between server and client is through socket programming.

2 Functionality

1. For the client program, there are two types of clients: admins and users. Admins have the functionality to add, remove and update products. Users have the functionality to view all the products being offered, add items to their cart and purchase all the items in their cart.
2. The server program has all file related functionalities. It reads from, writes to and locks the records in the file and also assigns customer ids to new customers.

3 Project File Structure

1. **structs.h** - Contains the struct definition of the Customer and Product struct and defines some macros used in other files.
2. **customers.dat** - Binary file containing a list of customers. It stores all the customer related information.
3. **products.dat** - Binary file containing a list of products. It stores all the products related information that was added by the admin.
4. **admin_logs.txt and transaction_logs.txt** - Contains a list of all transactions made by admins and customers respectively.
5. **intitialdata.c** - Initialises all the dat and txt files and writes customers with id -1 and products with id -1 into their respective dat files. This must be executed once, the first time before running the application.

6. **server.c** - Contains the code to run the server side and execute the server functionalities.
7. **admin.c and admin.h** - Contains the code that simulates working of the admin functionalities.
8. **user.c and user.h** - Contains the code that simulates working of the user functionalities.
9. **client.c** - Includes user.h and admin.h and allows a client to login as a user/admin and calls user and admin functions from their respective .c files.

4 Running the Project

1. First run the following commands to create all the data files:

```
$ cc initialdata.c -o initialdata
$ ./initialdata
```

   This will create all the data files and fill the .dat files with dummy data (all ids = -1)

2. Create two separate terminals:

   (a) In the first terminal, run this command to create the server file's executable:

```
$ cc server.c -o server
$ ./server
```

   If the message "Listening to connections" appears on the terminal, the server is running successfully.

   (b) In the second terminal, run this command to create the client file's executable:

```
$ cc client.c admin.c user.c -o client
$ ./client
```

   Then, follow the menu options to access the admin functionalities or the user functionalities.

5 Data Organisation

1. The Product struct has the following attributes:

   (a) **Id**: Unique product id assigned to each product

   (b) **Quantity** : Quantity of the product being offered by the admin

(c) **Price** : Price of the product

(d) **Name** : Name of the product (String)

The list of products are stored in the products.dat file.

2. The Customer struct has the following attributes:

   (a) **Customer_id**: Unique customer id assigned to each customer. 100 customers of ids from 1 to 100 are written to customers.dat file in initialdata.c

   (b) **Assigned**: Represents a boolean value. 1 means that the customer id of that customer struct has been assigned to a customer and 0 means that that customer id can still be assigned to a new customer.

   (c) **Cart_items**: An array of Products which represents the products that have been added to the cart of that customer's. The id of all the products in the array is initialised to -1 which means that a new product can be added at that location.

   The list of customers are stored in the customers.dat file.

6 Implementation of admin.c

1. **int setup_admin_connection()**:
   Sets up a socket using port number 5001 to connect with the server and writes a value of 0 to the server to indicate that an admin is trying to connect. Returns the socket file descriptor so that it can be used by other functions.

2. **void print_products(struct Product products[])**:
   Used to print all products. Uses the logic that if a product's id != −1 in a list of product structs, it is a valid product, i.e. all of its information can be printed.

3. **void admin_options(int sd)**:
   Provides all the admin functionalities in the form of a menu driven program. Accepts a 'choice' integer from the user and takes respective inputs from the user based on the option given. Writes all of the information given by the admin to the server through the 'sd' socket descriptor to allow the server to access the data files and perform the necessary actions.

   The menu options provided to an admin include:

   (a) Add a new product

   (b) Delete a product

   (c) Update the price of a product

   (d) Update the quantity of a product

   (e) See all the products being offered

7 Implementation of user.c

1. **int setup_user_connection()**:
   Sets up a socket using port number 5001 to connect with the server and writes a value of 1 to the server to indicate that a user is trying to connect. Returns the socket file descriptor so that it can be used by other functions.

2. **void display_products(struct Product products[])**:
   Used to print all products. Uses the logic that if a product's id != −1 in a list of product structs, it is a valid product, i.e. all of its information can be printed.

3. **void user_options(int sd)**:
   Provides all the user functionalities in the form of a menu driven program. Accepts a 'choice' integer from the user and takes respective inputs from the user based on the option given. Writes all of the information given by the user to the server through the 'sd' socket descriptor to allow the server to access the data files and perform the necessary actions.

   The menu options provided to a user include:

   (a) See all the products being offered

   (b) Add an item to the cart

   (c) View all items in the cart

   (d) Update the quantity of a product in the cart

   (e) Delete an item in the cart

   (f) Buy all items in the cart (Enter payment portal)

8 Implementation of client.c

1. client.c includes user.h and admin.h and hence has all the functions in user.c and admin.c

2. **int main()**:
   This is the entry point function when we run ./client. It asks the user if they want to login as an admin or a user.

   (a) If we login as a user, it asks the user if they are an existing customer or a new customer. If they are new, the server assigns a customer id to them by changing the 'assigned' variable of a customer struct to 1. If they are an existing customer, we input the customer_id from the user. Then, we setup the user connection which returns a socket file descriptor and pass the customer id to the server.
   We then call the **user_options(sd)** function from user.c to provide the user menu options.

(b) If we login as an admin, we set up the admin connection and get the socket file descriptor.
We then call the **admin_options(sd)** function from admin.c to provide the admin menu options.

9 Implementation of server.c

1. **int setup_connection()**:
Creates a socket file descriptor for the server.

```
int setup_connection(){
    struct sockaddr_in server;
    int sd;
    socklen_t clientLen;
    sd = socket(AF_INET, SOCK_STREAM, 0);

    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(5001);

    bind(sd, (struct sockaddr *)&server, sizeof(server));
    listen(sd, 50);
    return sd;
}
```

2. **int main()**:
Entry point for the server. Creates a concurrent server using fork() system call and accepts incoming client connections.

   First, it reads the type of client connecting to the server (user/admin).
   If a customer is connecting, it either assigns a customer id (for a new customer) or accepts the customer id (for an existing customer).

   Next, it reads the menu choice inputted by the client on the client side and calls the required function to serve the request.
   Based on the choice inputted, it takes the required inputs for that option. For example, If an admin wants to update the price of a product, we read the product id and the new price from the client side and pass the arguments to the required function.

   Finally, all the functions return an integer value to denote success/failure and return multiple values if multiple types of failures are possible. These return values are directly written to the client side and the appropriate message is printed on the client terminal.

3. **int add_product(struct Product product, int nsd)** :
   Accesses the products.dat file.

   - First checks if there is already a product with the same id as the product trying to be added. If there is we write -1 to the client.
   - Next, we check if there is a product with id as -1, which means we can write the product at that position, in the products.dat file. If there is, we write the new product record at that spot and write 1 to the client to indicate success.
   - If there is no free spot, we write -2 to the client to indicate that the maximum number of products have already been added.

4. **int delete_product(int id, int nsd)** :
   Accesses the products.dat file.

   - Finds the product in the products.dat file where the id is the same as the id passed from the admin.
   - Locks the record in the dat file, reads the record, updates the id of that record to -1 to indicate that the product has been deleted, and rewrites the record back to the file. We write the value of 1 to the client to indicate success.
   - If there is no such record in the dat file, we return a value of -1 to denote failure.

5. **int update_price(int id, int price, int nsd)** :
   Accesses the products.dat file.

   - Finds the product in the products.dat file where the id is the same as the id passed from the admin.
   - Locks the record in the dat file, reads the record, updates the price of that record to the new price and rewrites the record back to the file. We write the value of 1 to the client to indicate success.
   - If there is no such record in the dat file, we return a value of -1 to denote failure.

6. **int update_quantity(int id, int quantity, int nsd)** :
   Accesses the products.dat file.

   - Finds the product in the products.dat file where the id is the same as the id passed from the admin.
   - Locks the record in the dat file, reads the record, updates the quantity of that record to the new quantity and rewrites the record back to the file. We write the value of 1 to the client to indicate success.
   - If there is no such record in the dat file, we return a value of -1 to denote failure.

7. **void send_products(int nsd)** :
   Accesses the products.dat file.
   - We read all valid products from the products.dat file and write them to the client as an array and display them on the client side.

8. **int add_cart_item(int customer_id,int product_id,int quantity)** :
   Accesses the customers.dat file
   - Finds the customer in customers.dat file with id same as the customer that is accessing the server.
   - We go through the products in the cart_items array and find the first product with id as -1.
   - We update the id and quantity of the product to those passed by the user. (After checking that the product indeed exists and the quantity passed is $\leq$ than what is there in the products.dat file).

9. **int update_cart_item(int customer_id,int product_id,int quantity)**:
   Accesses the customers.dat file
   - Finds the customer in customers.dat file with id same as the customer that is accessing the server.
   - We go through the products in the cart_items array and find the product with id same as the id passed by the user.
   - We update the quantity of the product to that passed by the user. (After checking that the quantity passed is $\leq$ than what is there in the products.dat file).

10. **int delete_cart_item(int customer_id,int product_id)**:
    Accesses the customers.dat file
    - Finds the customer in customers.dat file with id same as the customer that is accessing the server.
    - We go through the products in the cart_items array and find the product with id same as the id passed by the user.
    - We update the product id of that product to -1, to indicate that there is not valid product at that location.

11. **void send_cart_items(int customer_id, int nsd)**:
    Accesses the customers.dat file
    - Finds the customer in customers.dat file with id same as the customer that is accessing the server.
    - We write the cart_items array to the client and print the valid cart items on the client side (using void display_products()).

12. **void payment_portal(int user_id, int nsd)**:
    Accesses the customers.dat file and the products.dat file

    - Finds the customer in customers.dat file with id same as the customer that is accessing the server.
    - Finds a valid product in the customer's cart and locks the respective product in the products.dat file (if it exists).
    - Checks if the quantity in the cart of the customer is $\leq$ the quantity available in the products.dat file. If it is, we decrease the quantity in the product.dat file by the amount of that product in the customer's cart. If it isn't we ignore that product.
    - We repeat this for all the valid items in the customer's cart.
    - We also clear the customer's cart by by setting all product ids as -1 and we re-write the customer struct back to the customers.dat file in the same position.

10 OS Concepts Used

1. **Socket Programming** - To communicate between client and server and to abstract the server implementation away from the client.
2. **open, read, write System Calls** - To access the data files and perform access operations on them.
3. **File Locking** - To ensure data concurrency between admins and clients.