SYSTEM LAB ASSIGNMENT REPORT

GROUP NO -21

Banking system using Client-Server socket programming

Submitted by -
Kishore M -214101062
Divyanshu Nauni -214101063
Rishab Deo Singh-214101064

Banking application is done using socket programming to show how transactions are done when different types of client trying to acess the banking system.

Main aim of this project is to develop a online banking software application through which basic functionality like deposit, balance enquiry, managing account settings, client withdrawal functionality can be performed. Networking banking system can also be called as online baking. This application works on two tire architecture, bank clients and bank server. Here we provide full project code for client and server with database.

LOGIN file will store the client number , client username and password .
A text file will be maintained corresponding to each username to maintain the statement of the account. Currently we have 10 customer credentials in login file.

Once the Bank server receives the login request, it validates
the information and performs the functionalities according to the user mode type.

Entities in Banking system and their functionality and access privillaeges :-

- Bank_Customer: The customer should able to see AVAILABLE BALANCE in his/her account and mini statement of his/her account.

- Bank_Admin: The admin should be able to CREDIT/DEBIT the certain amount of money from any Bank_Customer ACCOUNT . The admin must update the respective “Customer_Account_file” by appending the new information. Handle the Customer account balance underflow cases carefully.

- Bank_Police: The police should only be able to see the available balance of all customers and MINI STATEMENT by quoting the Customer _ID (i.e. User_ID with user_type as ‘P’).

Code bits used for these entities are named as:-

1. Server.c – will handle verifying credentials , will bind clients to server , will fork to handlemultiple clients . Will call service handler by passing client fd .
2. Client.c - will take input from user and write into buffer, initiate connection establishment with server , and will call respective helper client function depending upon verifying credential response it will get from server.

3. Client_admin -- Client of admin acess will perform transaction will be recorded via communication between client admin and server admin .

4. Server_Admin -The admin should be able to CREDIT/DEBIT the certain amount of money from any Bank_Customer ACCOUNT . The admin must update the respective “Customer_Account_file” by appending the new information. Handle the Customer account balance underflow cases carefully. So server_admin.c will contain respective functions to implement these.

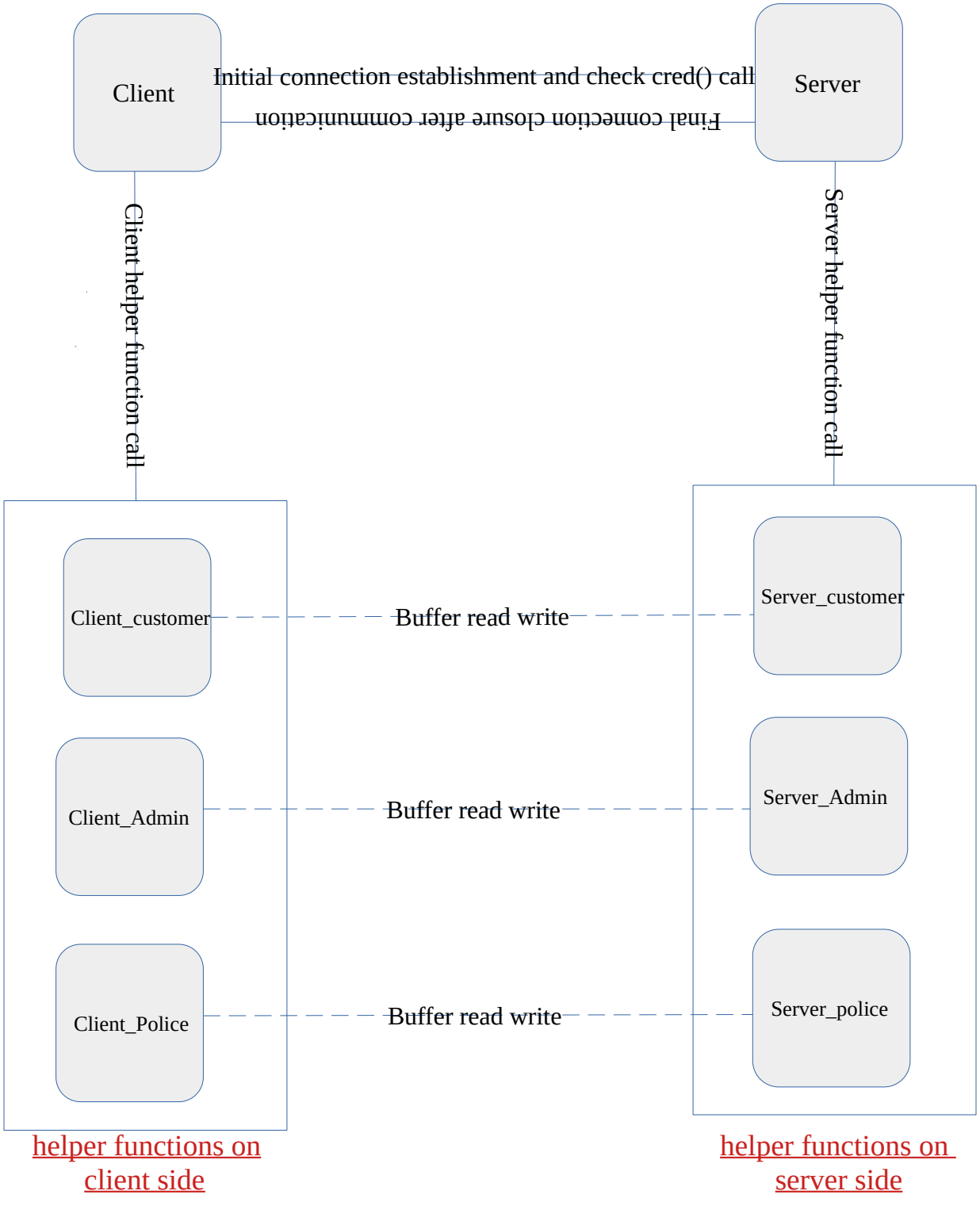
5. client_police.c - client of police acess will perform transaction will be recorded via communication between client police and server police . Control will be transfered as passed by client.c program for handling client of police type by providing options .

6. Server_police - The police should only be able to see the available balance of all customers. He is allowed to view any Customers MINI STATEMENT by quoting the Customer _ID (i.e. User_ID with user_type as ‘C’). So server_admin.c will contain respective functions to implement these and will be called upon as it reads instructions from buffer.

7. Client_customer.c - client of customer acess will perform transaction will be recorded via communication between client customer and server customer. Control will be transfered as passed by client.c program for handling client of customer type by providing options as parametre.

8. server_customer - The customer should able to see AVAILABLE BALANCE in his/her account and mini statement of his/her account. So server_admin.c will contain. respective functions to implement these and will be called upon as it reads instructions from buffer.

flow for the client server program -



Read() is blocking read .
Write() is non blocking .

FLOW Explanation -

Client.c will estbalish a connection with server.h and will pass the credentials taken as input from user following which server will call Login() function which will in turn call Check_cred function which will give three attempt (at max) to read credentials from buffer to check client type and provide response back to client.c .

Following which , corresponding to client type , server will call helper server function and client will call counterpart client helper function .

After this step communication will take place in between 2 helper functions (available for each client type for example : for admin client type will be namely client_admin.c and server_admin.c where server which will contain functions representing functionalities of that client).

And finally connection closure will take place.

Some important inbuilt Methods and used in socket programming :-

Step 1 – Setup Socket

• Both client and server need to setup the socket

– int socket(int domain, int type, int protocol);

• **Domain**
– AF_INET -- IPv4 (AF_INET6 for Ipv6)

• **Type**

– SOCK_STREAM --TCP
– SOCK_DGRAM ---UDP

• **protocol**
– 0
• For example,
– int sockfd = socket(AF_INET, SOCK_STREAM, 0);

Step 2 (Server) --- Binding

• **Only server need to bind**
– int bind(int sockfd, const struct sockaddr *my_addr,

socklen_t addrlen);
• **sockfd**
– file descriptor socket() returned
• **my_addr**
– struct sockaddr_in for IPv4
– cast (struct sockaddr_in*) to (struct sockaddr*)

```
struct sockaddr_in {
short sin_family; // e.g. AF_INET
unsigned short sin_port; // e.g. htons(3490)
struct in_addr sin_addr; // see struct in_addr, below
char sin_zero[8]; // zero this if you want to
};
struct in_addr {
unsigned long s_addr; // load with inet_aton()
}
```

Step 3 (Server) - Listen

• Now we can listen
– int listen(int sockfd, int backlog);
• **sockfd**
– again, file descriptor socket() returned
• **backlog**
– number of pending connecCons to queue
• For example,
– listen(sockfd, 5)

Step 4 (Server) - Accept

• **Server must explicitly accept incoming connections**
– int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen)
• **sockfd**
– again... file descriptor socket() returned
• **addr**
– pointer to store client address, (struct sockaddr_in *) cast to (struct sockaddr *)
• **addrlen**
– pointer to store the returned size of addr, should be sizeof (*addr)
• For example
– int isock=accept(sockfd, (struct sockaddr_in *) &caddr, &clen);

All client need to do is to connect

– int connect(int sockfd, const struct sockaddr *saddr, socklen_t addrlen);
• For example,
– connect(sockfd, (struct sockaddr *) &saddr, sizeof (saddr));