4/6/2015

Kodjo Komedja, Durko Justin | Computer Networks | April 06 2015

**UNIVERSITY OF NORTH TEXAS**

**PROJET**

Bank ATM Machine using CNP\_Protocol

**Table of Contents**

1. Problem being solved
2. Approach
3. Implementation
4. **Problem being solved**

This project states a concept see in application layer protocol we cover in class and is based on the best Application Protocol Design during group activity. Our goal is to implement a Client and Server application with a new protocol which will be a software for the next generation of ATM machines. The software features are

Creating account: (first and last names, pin, DL, SSN, and email address)

Authentication of a valid user (first name and pin)

Deposit (resolve problem of the bolt in the machine is full)

Withdraw (resolve problem of the bolt in the machine not having enough money)

Verification of action

Balance

Show last n transactions

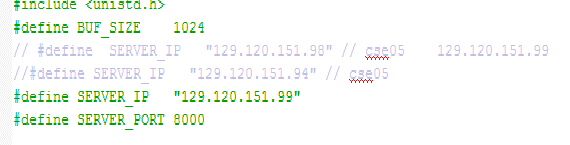
Buy stamps

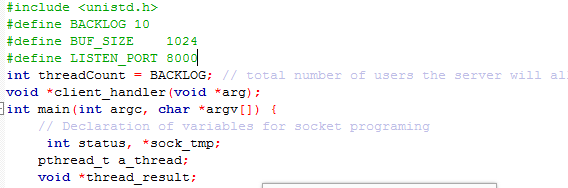
Terminate the transaction

1. **Approach to solve the problem**

**Understanding socket programing**

On client side, we start by defining server address and port number which are important to know, so we can change it inside in code depending on the server and the port number available.



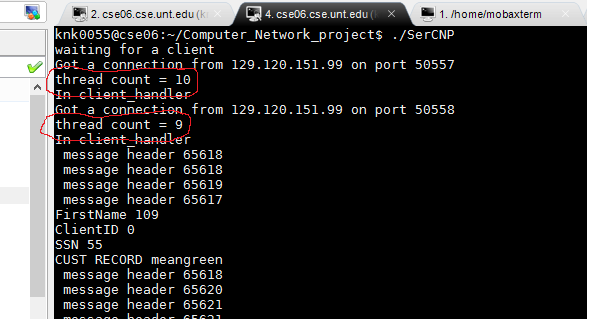
On Server side we make sure it is the same defined port number LISTEN\_PORT

There are followed by defined variables int sock-send,sockaddr\_in, addr\_send which are required to initialize a socket end point connection and to make the binding.

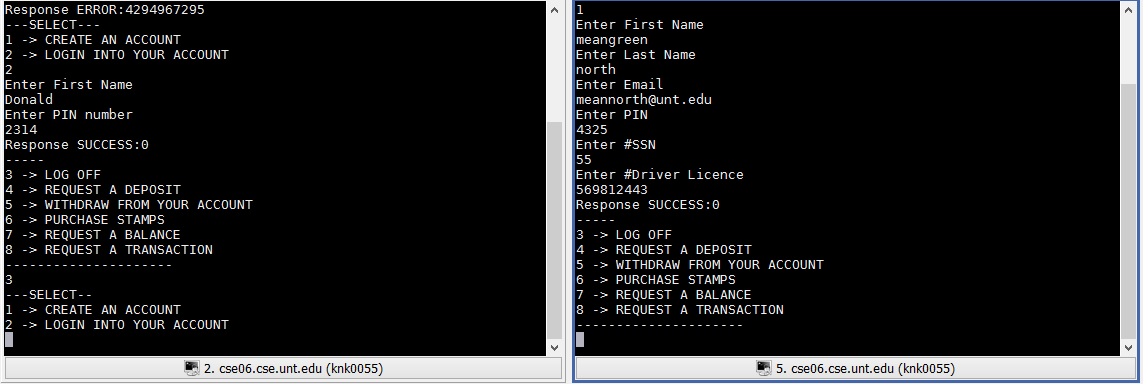
In client side the socket is created with sock\_send=socket(PF\_INET,SOCK\_STREAM,…)

We connect to server socket with connect(sock\_send, &addr\_send,….)

**Understanding multithread in C++**

On server side only, required variables are pthread\_t , thread\_result. We use a socket binding same as on client side, and we listen to port number from client application and launch a thread to handle the client operation. A threadcount is initialized to 10 and decrement each time a new client is connected (which means our server can only accept 10 users) which is enough for multithreading clients-server. 

Another example of multithreading (two clients connected server)



**Understanding writing data to file as ofstream and reading from file as ifstream**

This allows us the capability to save our data in file which is very useful for Create account request, Authentication, Balance request, withdrawal request… Because the client/server application is executed on cse06.cse.unt.edu (UNT computer machines) which don’t have a preinstalled database available, we can only save our data in file. We will clarify step by step how our code work in next following lines.

**Understanding CNP\_Protocol.h data structure**

* Creating account is defined by a data structure CREATE\_ACCOUNT\_REQUEST and CREATE\_ACCOUNT\_RESPONSE
* Authentication into account is defined with a data structure LOGON\_REQUEST and LOGON\_RESPONSE
* Deposit is achieved with a data structure DEPOSIT\_REQUEST and DEPOSIT\_RESPONSE
* Withdraw is accomplished with data structure WITHDRAWAL\_REQUEST and WITHDRAWAL\_RESPONSE
* Balance is defined by data structure BALANCE\_QUERY\_REQUEST and BALANCE\_QUERY\_RESPONSE
* Show last n transactions is done with TRANSACTION\_QUERY\_REQUEST and TRANSACTION\_QUERY\_RESPONSE
* Buy stamps STAMP\_PURCHASE\_REQUEST and STAMP\_PURCHASE\_RESPONSE

**Additionnally DataMap.h data structure**

Define a data structure for clients records in file.

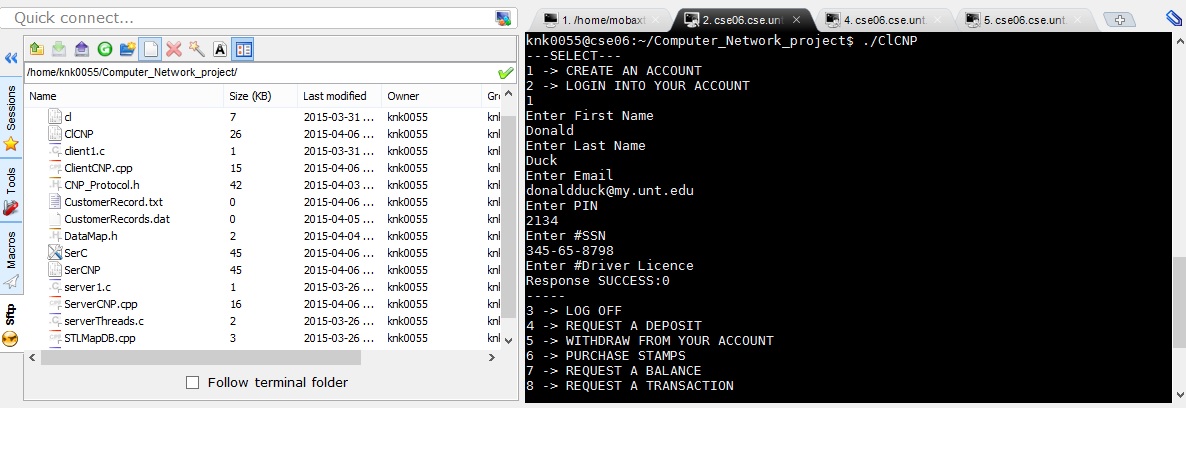
1. **Implementation**

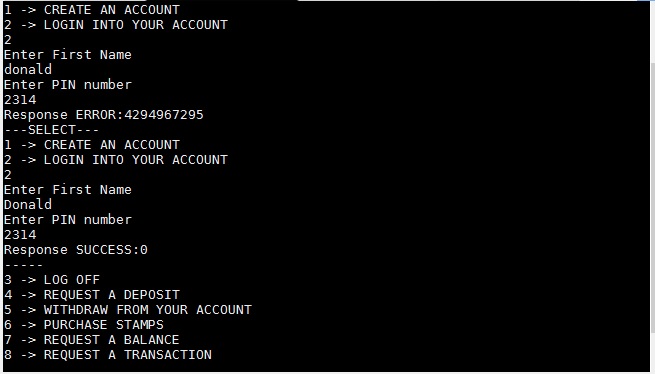
**Client side code**

A default constructor CONNECT\_REQUEST (refers to CNP\_Protocol.h) is created . this message is sent to server. If an error connection occurs, an ERROR\_CONNECT\_RESPONSE is issued. Otherwise, we receive acknowledgment of success and a main menu is displayed to client with two option

1 -> CREATE AN ACCOUNT

2 -> LOGIN INTO YOUR ACCOUNT

When user types 1 as entry the message protocol CREATE\_ACCOUNT\_REQUEST is defined , then the client is asked to “enter first name” , “enter last name”,”pin number” etc.. which are collected as parameters for the constructor CREATE\_ACCOUNT\_REQUEST then sent to server. If the request is successful a response from CREATE\_ACCOUNT\_RESONSE is sent from server to client which contains CER\_SUCCESS (means successful) ,otherwise CER\_ERROR(means error) 

When user select 2 , he will be asked to verify his credentials with first name and Pin number ,then a LOGON\_REQUEST is initiated with its parameters including first name and pin number sent to server side. With correct key, the user is authorized to access to his account in order to select the type of transaction he wants. 

If authentication failed, he will be asked again to type his credentials.

After creating an account or login successfully the client has account menu which looks like this

3 -> LOG OFF

4 -> REQUEST A DEPOSIT

5 -> WITHDRAW FROM YOUR ACCOUNT

6 -> PURCHASE STAMPS

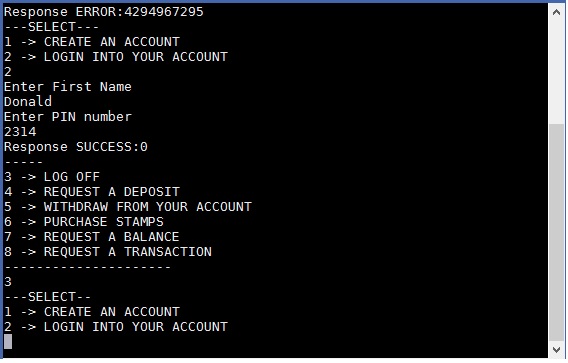
7 -> REQUEST A BALANCE

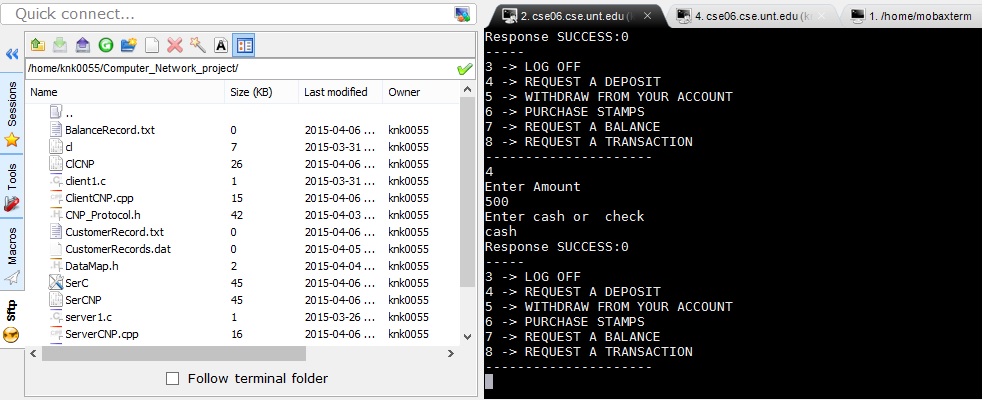
8 -> REQUEST A TRANSACTION

When a client select 3, an LOGOFF\_REQUEST message protocol is issued and sent to server who redirects the user to main menu with LOGOFF\_RESONSE

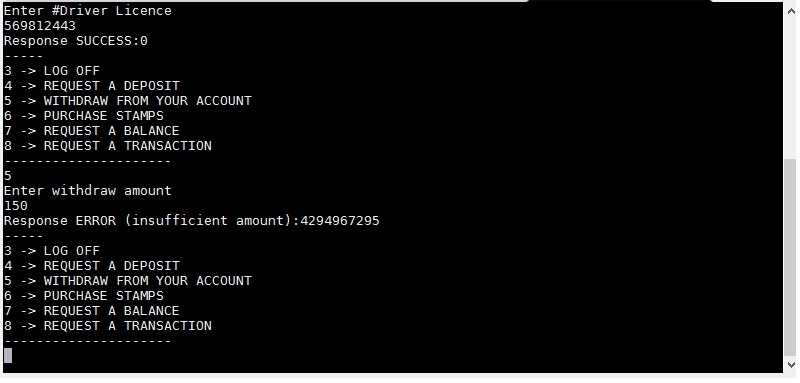
1 -> CREATE AN ACCOUNT

2 -> LOGIN INTO YOUR ACCOUNT

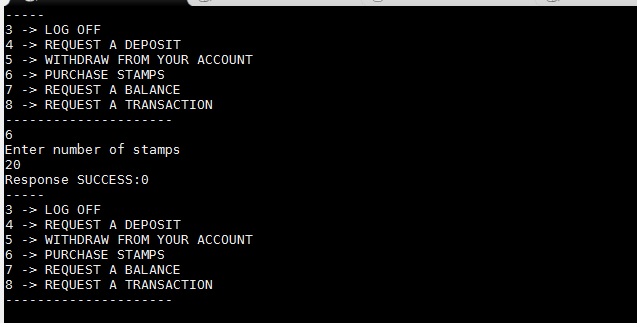


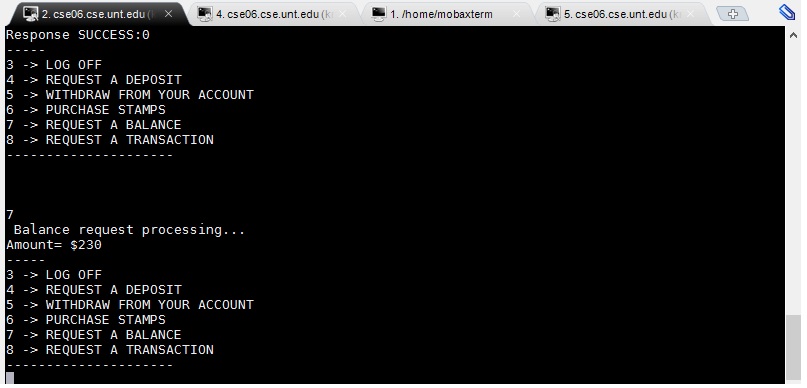
When a user select 4, DEPOSIT\_REQUEST structure protocol is defined , then the client is prompted to “enter amount”, “enter cash or check”. The request is sent to server with a valid MT\_DEPOSIT\_ID for getting information and execute instructions on server side before getting back the response DEPOSIT\_RESPONSE data structure which contains the type of result CER\_SUCCESS or CER\_ERROR

Just for information a BalanceRecord.txt is created if it is the first user to make a deposit or rewritten if already exist. This balanceRecord.text contains the ClientID which is passed from Authentication protocol or Create account protocol to others message protocols belonging to Account menu.

When a user request a WITHDRAWAL\_REQUEST message protocol, the request protocol is sent to server which looks through BalanceRecord.txt to check user’s amount greater or equal to amount requested. If the comparison is false an response of insufficient amount is issued to client side through WITHDRAWAL\_RESPONSE protocol. 

When the client choose 6 , then a STAMP\_PURCHASE\_REQUEST default constructor is defined , then the customer will enter the number of stamps which with ClientID are assigned to the constructor for re-initialization and therefore, sent to server for processing request .A response STAMP\_PURCHASE\_RESPONSE is sent back from server to client to confirm the result of the request.



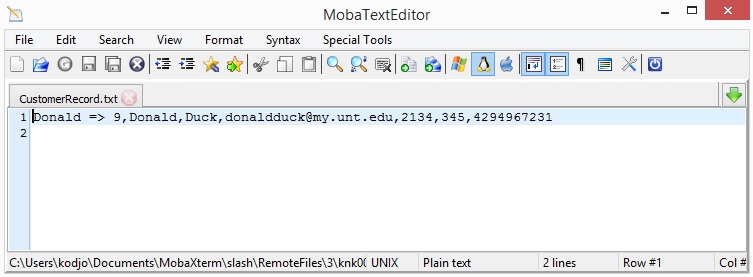
When a user opts for 7, a BALANCE\_QUERY\_REQUEST is defined, and the request is sent to server. The server check in BalanceRecord.txt the amount corresponding to ClientID and then replies back to client side with a response protocol BALANCE\_QUERY\_RESPONSE which displays the amount

Finally, when a customer, selects 8, a TRANSACTION\_QUERY\_REQUEST protocol is initialized allowing the user to enter TRANSACTION\_ID and number of transaction. the result will come from the server which returns TRANSACTION\_QUERY\_RESPONSE protocol.

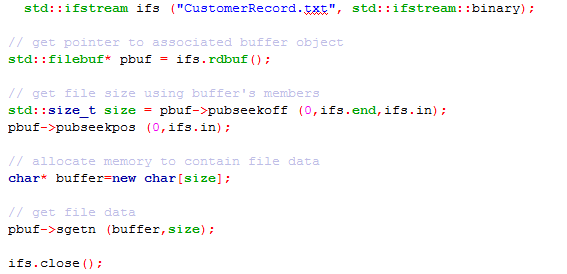
**Server side code**

When a CREATE\_ACCOUNT\_REQUEST protocol is sent from client, the server gets the data from the data structure and save the information in file CustomerRecord.txt which is created for a the first client and added for new users

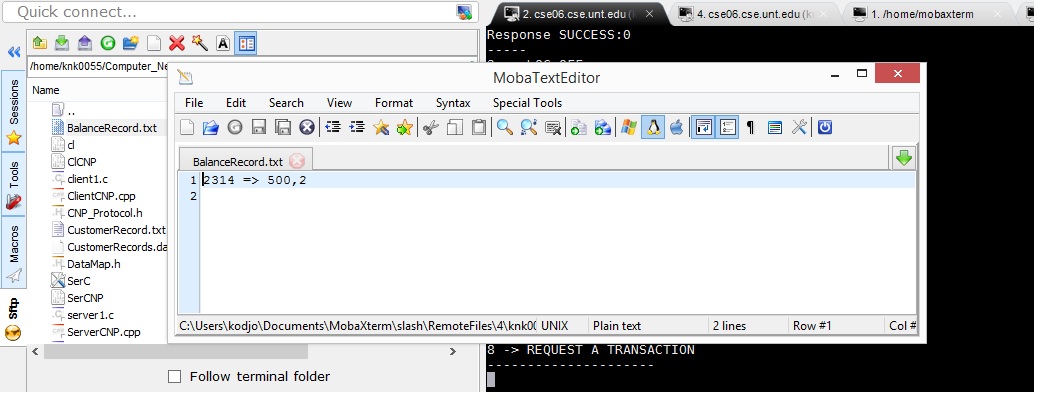
.



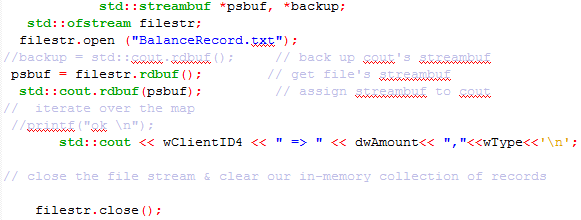
When LOGON\_REQUEST protocol is called, the server with the client key read the CustomerRecord.txt to check if this user exist in the file. Then, then the server initiate a response protocol LOGON\_RESPONSE back to client



For BALANCE\_QUERY\_REQUEST protocol, the server with the clientID read through BalanceRecord.txt to get the amount which will be sent through a response protocol BALANCE\_QUERY\_RESPONSE



For DEPOSIT\_REQUEST protocol, the server creates a BalanceRecord.txt (if first user) or update BalanceRecord.txt (if new users )



At the end, the server send a notification response DEPOSIT\_RESPONSE back to client, with MT\_DEPOSIT\_ID, and clientID.