Introduction to Computer Network:
Lab1

1. Overview:
   In this lab, we are required to implement a transaction model. We could take it as a banking system. In the banking system, the client can connect with the server site and make transaction. And, the server should handle the connection from the client (user).

   Besides, for the server, it only need to handle one connection from the client, which means we do **not need to use fork() or thread()** to implement a concurrent server.

   Moreover, the entire system runs on windows environment. Thus, winsock API is required instead of socket for linux. Fortunately, they are only slightly different. The foundation concept is same. And then, the system is implemented via TCP
   It runs in C++11

2. Architecture:
   For the architecture design, it can be decomposed into 2 parts: one is for the client ,and the other is for server.
   1. Client side:
      i. 103011227_cli.cpp
         1. It connects and sends command to server.
         2. Basically, it performs socket(),connect(),recv(),send(),closesocket() socket operations.
   2. Server side:
      i. 103011227_ser.cpp
         1. It receive connection from client and parse the commands to perform corresponding service.
         2. Fundamentally,it performs socket(),bind(),listen(),accept(),recv(),send(),closesocket(),socket operation
         3. It supports
            1. Login(l): login the account

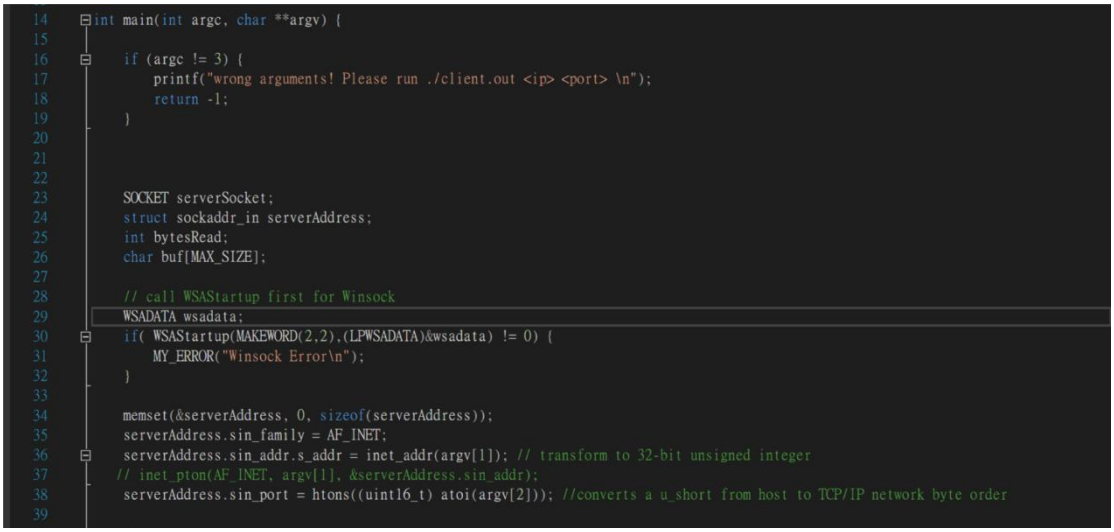
2. Create(c): create a new account
3. Deposit(d): deposit the money into the bank
4. Withdraw(w): withdraw money from user's account
5. Check Info(i): check user's information
6. Exit(exit): leave the system

ii. Account.h
1. It defines the data structure of the account

iii. BankSystem.h
1. It defines the bank system to manage accounts and store the data to BankData.txt

iv. BankData.txt
1. It is the simple database for the BankSystem to store the entire data.

3. Detailed Design:
1. Client Side:
i. 103011227_cli.cpp



```
int main(int argc, char **argv) {

    if (argc != 3) {
        printf("wrong arguments! Please run ./client.out <ip> <port> \n");
        return -1;
    }



    SOCKET serverSocket;
    struct sockaddr_in serverAddress;
    int bytesRead;
    char buf[MAX_SIZE];

    // call WSAStartup first for Winsock
    WSADATA wsadata;
    if( WSAStartup(MAKEWORD(2,2),(LPWSADATA)&wsadata) != 0) {
        MY_ERROR("Winsock Error\n");
    }

    memset(&serverAddress, 0, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = inet_addr(argv[1]); // transform to 32-bit unsigned integer
    // inet_pton(AF_INET, argv[1], &serverAddress.sin_addr);
    serverAddress.sin_port = htons((uint16_t) atoi(argv[2])); //converts a u_short from host to TCP/IP network byte order

```

In the beginning, we get the argument parameter from the command line. Then, we set up the socket to connect the server

```
serverSocket = socket(PF_INET, SOCK_STREAM, 0);
if(connect(serverSocket, (struct sockaddr *)&serverAddress, sizeof(serverAddress))<0){
    MY_ERROR("Connect Error\n");
}

bytesRead = recv(serverSocket, buf, MAX_SIZE, 0);
printf("Receive %d byte(s): %s\n", bytesRead, buf);

while (scanf("%s", buf)>0) {
    if (!strcmp("exit", buf)) {
        send(serverSocket, buf, strlen(buf), 0);
        closesocket(serverSocket);
        return 0;


    }

    //write(sockfd, sendline, strlen(sendline));
    send(serverSocket, buf, strlen(buf), 0);
    bytesRead = recv(serverSocket, buf, MAX_SIZE, 0);
    buf[bytesRead] = '\0';
    if (bytesRead>0) printf("------------------------------\nReceive %d byte(s): %s\n", bytesRead, buf);

    if (!strcmp(buf, "Bye Bye")) {
        closesocket(serverSocket);
        return 0;
    }

    /*
```

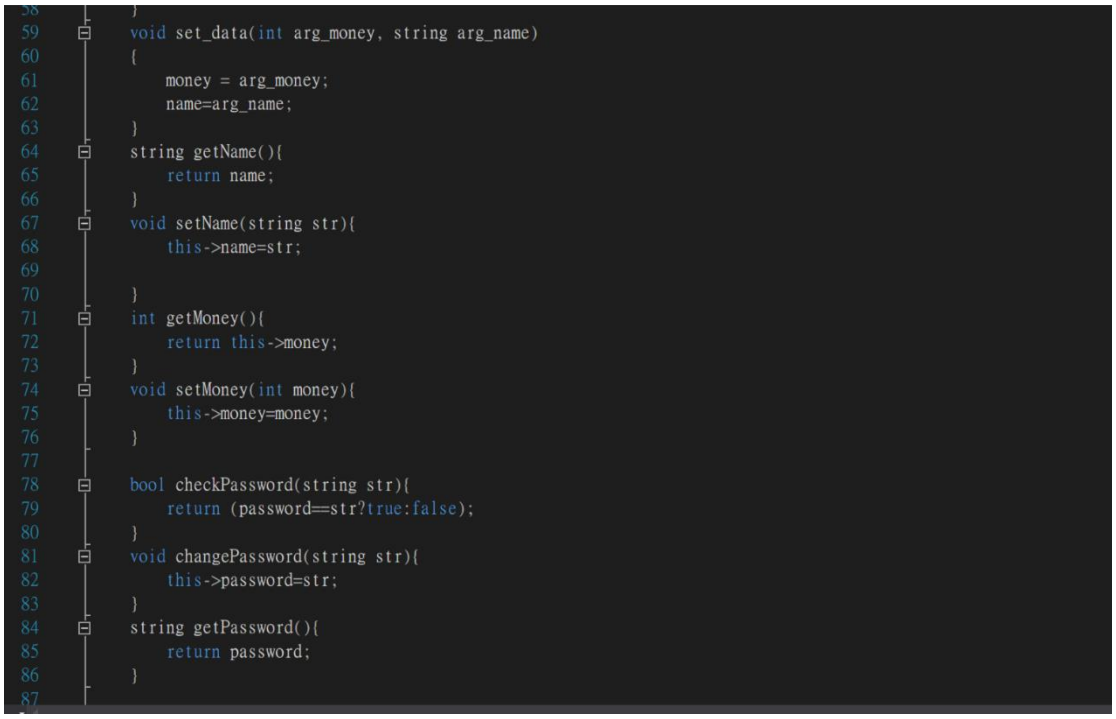
After connecting the server, it is implemented as an echo client/server. Besides, if the user type “exit”, it will send the message to notify the server and shut down client’s process.

In brief, in this client part, we make a slight modification of echo client.

2. Server Side:
   i. Account.h: It defines the data structure for each account

```
#ifndef Account_h
#define Account_h

class Account {
private:
    int money;
    string name;
    string password;
public:
    Account() { ... }
    Account(string name) { ... }
    Account(string name, int money) { ... }
    Account(string name, int money,string code){
        this->name=name;
        this->money=money;
        this->password=code;
    }


    void get_data()
    {
        cout << money << endl;
        cout << name << endl;
    }
    void set_data(int arg_money, string arg_name)
    {
        money = arg_money;
        name=arg_name;
```

For private field, it contains **user's money, name, and password**
And we defined multiple constructors to instantiate the object.



```
59  void set_data(int arg_money, string arg_name)
60  {
61      money = arg_money;
62      name=arg_name;
63  }
64  string getName(){
65      return name;
66  }
67  void setName(string str){
68      this->name=str;
69
70  }
71  int getMoney(){
72      return this->money;
73  }
74  void setMoney(int money){
75      this->money=money;
76  }
77
78  bool checkPassword(string str){
79      return (password==str?true:false);
80  }
81  void changePassword(string str){
82      this->password=str;
83  }
84  string getPassword(){
85      return password;
86  }
87
```

Additionally, it provides **set and get** operation to set and get the data. This is because the data is defined as private and this way can offer **data protection** and better coding style.

Also, checkPassword() can check whether the input string matches correct password and changePassword() could change user's passwords.

ii. BankSystem.h: it is the system that manages the account and store the account information in BankData.txt.

```
class BankSystem {
private:
    Account write_data,read_data;
    vector<Account>list;

public:
    BankSystem(){


        ifstream infile;
        infile.open ("BankData.txt", ios::in|ios::binary);
        string name;
        int money;
        string password;
        while (infile>>name>>money>>password ) {
                Account account(name,money,password);
                list.push_back(account);
        }
        infile.close();


   /*
        FILE *file = fopen("BankData.txt", "rb");
        if (file != NULL) {

            while (fread(&account, sizeof(account), 1, file) != 0) {
                list.push_back(account);
            }
```

For private field, it contains **Account write_data,read_data,** and **vector<Account>list.** Plus, write_date,read_data are for read and write data and are abandoned in current version. What's more, **list** contains all the accounts with their information in the bank system.

Whenever the system is instantiated, it **loads the entire account data from BankData.txt into the list** by push_back()

```
61  void addCount(string name, int money,string code){
62      Account tmp=findAccount(name);
63      if(tmp.getName()!=name){
64          tmp.get_data();
65          list.push_back(Account(name,money,code));
66      }
67  }
68
69  Account findAccount(string name){
70      for(auto c:list ){
71          if(c.getName()==name){
72              cout<<"Accout exist"<<endl;
73              return c;
74          }
75      }
76      return Account("NULL");
77  }
78
79  bool authenticate(string name , string code){
80      Account tmp=findAccount(name);
81      if(tmp.getName()==name){
82
83          if(tmp.checkPassword(code)==true)
84              return true;
85          else
86              return false;
87      }
```

Also, the system support many functions required.

- addCount():
  - it add new account to the list if its name doesn't repeat
- findAccount():
  - it traverse the entire list to see if the corresponding account by name exist. If so, return the account. Otherwise, it returns an empty account
- authenticate():
  - it check the input name and code with the corresponding account's password. If the code match the password, the authentication succeed and return true. On the contrary, if it doesn't, the authentication fails and return false.

```
93      int setMoney(string name, int money){
94
95          for (auto i = list.begin(); i != list.end(); i++)
96          {
97              if(i->getName()==name){
98                  i->setMoney(money);
99                  return money;
100             }
101
102
103         }
104
105         return -1;
106
107     }
108
109     int deposit(string name,int money){
110         for (auto i = list.begin(); i != list.end(); i++)
111         {
112             if(i->getName()==name){
113                 int cur=i->getMoney();
114                 i->setMoney(cur+money);
115                 return cur+money;
116             }
117
118
119         }
120
121         return -1;
122
```

- setMoney():
  - it sets the money to the corresponding account. Usually, it is called when a new account it created
- deposit():
  - it deposit the money to the corresponding account.
  - If the account does exist, it add the input money to the money in the account, and it returns the remaining money after depositing.
  - Otherwise, it returns -1 as error.

```
    int withdraw(string name,int money){
        for (auto i = list.begin(); i != list.end(); i++)
        {
            if(i->getName()==name){
                int cur=i->getMoney();
                if(cur<money)return -1;
                i->setMoney(cur-money);
                return cur-money;
            }


        }

        return -1;

    }

    void print(){
        cout<<"Client Data"<<endl;
        cout<<"--------------------------"<<endl;
        for (int i=0; i<list.size(); i++) {
            cout<<list[i].getName()<<"--"<<list[i].getMoney()<<endl;

        }



    }
```

- withdraw():
  - It withdraw the money from the corresponding account.
  - It checks whether there is enough money in the account.
    - If there is enough money, user can withdraw the money and the system returns remaining money.
    - Otherwise, it returns -1 as error
- Print():
  - It traverse the entire list and print their information
  - It is for debugging purpose.

```
152     }
153
154
155     void update(){
156
157         fstream outfile;
158         outfile.open("BankData.txt",ios::binary|ios::trunc|ios::out);
159
160
161         for(auto c:list){
162             outfile<<c.getName()<<' '<<c.getMoney()<<' '<<c.getPassword()<<' ';
163         }
164         outfile.close();
165
166
167
168     }
169
170
171
172
173
```

- update():
  - It store the current list data into the Bankdata.txt
  - It is called when a user exit a system to update the corresponding data in the BankData.txt

iii. 103011227_ser.cpp: It is the main function for the bank system. Moreover, it is designed based on an echo server with modification.

```
12  using namespace std;
13
14  #define MAX_SIZE 2048
15  #define MY_ERROR(s) printf(s); system("PAUSE"); exit(1);
16  #define SERVER_PORT 9999
17
18  int main(int argc, char **argv)
19  {
20      SOCKET serverSocket, clientSocket; // create a socket
21      struct sockaddr_in serverAddress, clientAddress; // sockaddr_in：IP4 格式使用 ， sockaddr_in6：IP6 格式使用
22      int clientAddressLen;
23      int bytesRead;
24      char buf[MAX_SIZE];
25
26      stringstream ss;
27
28
29
30
31
32      if (argc != 2)
33      {
34          printf("wrong arguments! Please run ./server.out <port> \n");
35          return -1;
36      }
37      else
38      {
39          cout<<"port"<<atoi(argv[1])<<endl;
40      }
41
42
```

```
43      // call WSAStartup first for Winsock
44      WSADATA wsadata;
45      if( WSAStartup(MAKEWORD(2,2),(LPWSADATA)&wsadata) != 0)   // ( version of winsock )
46      {
47          MY_ERROR("Winsock Error\n");
48      }
49
50      serverSocket = socket(PF_INET, SOCK_STREAM, 0); // (address , type , protocal(0表示不強制) )
51
52      memset(&serverAddress, 0, sizeof(serverAddress));
53      serverAddress.sin_family = AF_INET;
54      serverAddress.sin_addr.s_addr = INADDR_ANY;
55      serverAddress.sin_port =  htons((uint16_t) atoi(argv[1]));//converts a u_short from host to TCP/IP network b
56
57
58
59      if( bind(serverSocket, (struct sockaddr *)&serverAddress, sizeof(serverAddress)) < 0)
60      {
61          MY_ERROR("Bind Error\n");
62      }
63
64      if( listen(serverSocket, 3) < 0)
65      {
66          MY_ERROR("Listen Error\n");
67      }
68
69
70
```

First of all, we set up the socket by the given argument from the command line. If there is anything wrong with the argument such as not enough arguments, it will prompt error message and exit.

Next, we perform bind(),listen(), accept() recv(),send(),closesocket() socket operations to construct

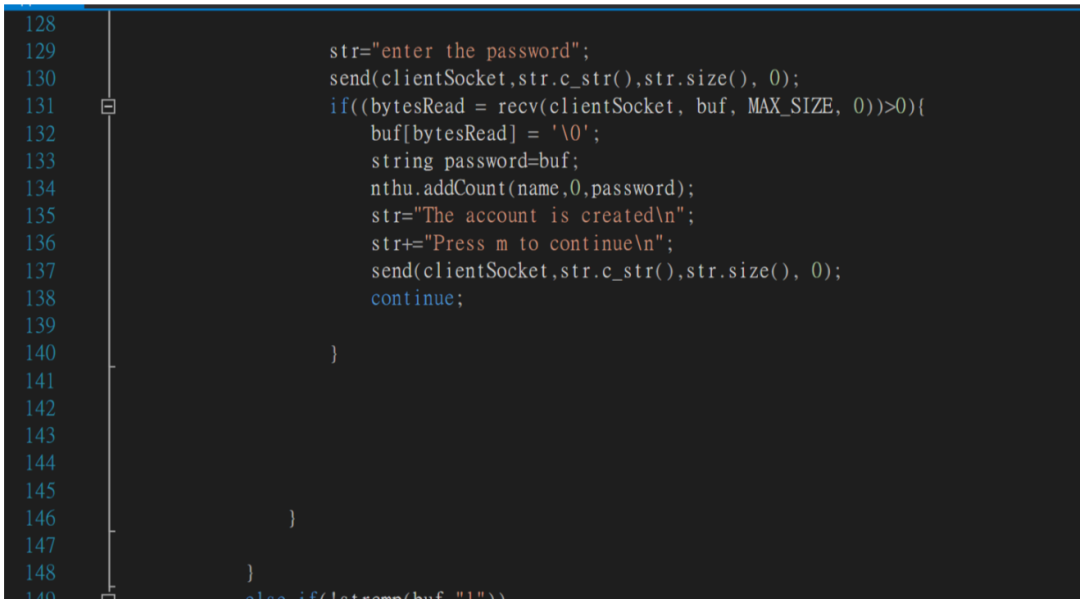
the whole transaction system.

```
while(1)
{
    printf("Waiting...\n");
    clientAddressLen = sizeof(clientAddress);
    clientSocket = accept(serverSocket, (struct sockaddr *)&clientAddress, &clientAddressLen);
    printf("Client IP is : %s \n", inet_ntoa(clientAddress.sin_addr));
    string command;
    command="\nwelcome to Bank";
    command+="\nlogin(l)";
    command+="\ncreate(c)\n";
    send(clientSocket,command.c_str(), command.size(), 0);

    command="\n-----------------------------\n";
    command+="\nlogin(l)";
    command+="\ncreate(c)\n";
    command+="exit(exit)\n";


    string user_name;



    again:

    BankSystem nthu;

```

After the user enters the system, the server side will show client's IP address send welcome message as well as menu to him.

Moreover, it will instantiate an object of BankSystem, called nthu, to manage the account system.

```
    while( (bytesRead = recv(clientSocket, buf, MAX_SIZE, 0))>0  )
    {
        buf[bytesRead] = '\0';
        if(!strcmp(buf,"exit"))
        {
            nthu.update();
            cout<<"Client Want to leave"<<endl;
            send(clientSocket,"Bye Bye",sizeof("Bye Bye"),0);
            command="\nwelcome to Bank";
            command+="\nlogin(l)";
            command+="\ncreate(c)\n";
            break;
        }
        else if(!strcmp(buf,"c"))
        {
            string str="enter your name";
            send(clientSocket,str.c_str(),str.size(), 0);
            printf("send %d byte(s) to user to ask for the account name: %s\n",str.size(),str.c_str());

            if((bytesRead = recv(clientSocket, buf, MAX_SIZE, 0))>0){
                cout<<"recieve user account name "<<buf<<endl;
                buf[bytesRead] = '\0';
                string name=buf;

                if(nthu.findAccount(name).getName()==name){
                    str="The account name already exist\n";
                    str+="Press m to continue\n";
                    send(clientSocket,str.c_str(),str.size(), 0);
                    continue;
                }
```



```
128
129                     str="enter the password";
130                     send(clientSocket,str.c_str(),str.size(), 0);
131                     if((bytesRead = recv(clientSocket, buf, MAX_SIZE, 0))>0){
132                         buf[bytesRead] = '\0';
133                         string password=buf;
134                         nthu.addCount(name,0,password);
135                         str="The account is created\n";
136                         str+="Press m to continue\n";
137                         send(clientSocket,str.c_str(),str.size(), 0);
138                         continue;
139
140                     }
141
142
143
144
145
146                 }
147
148             }
```

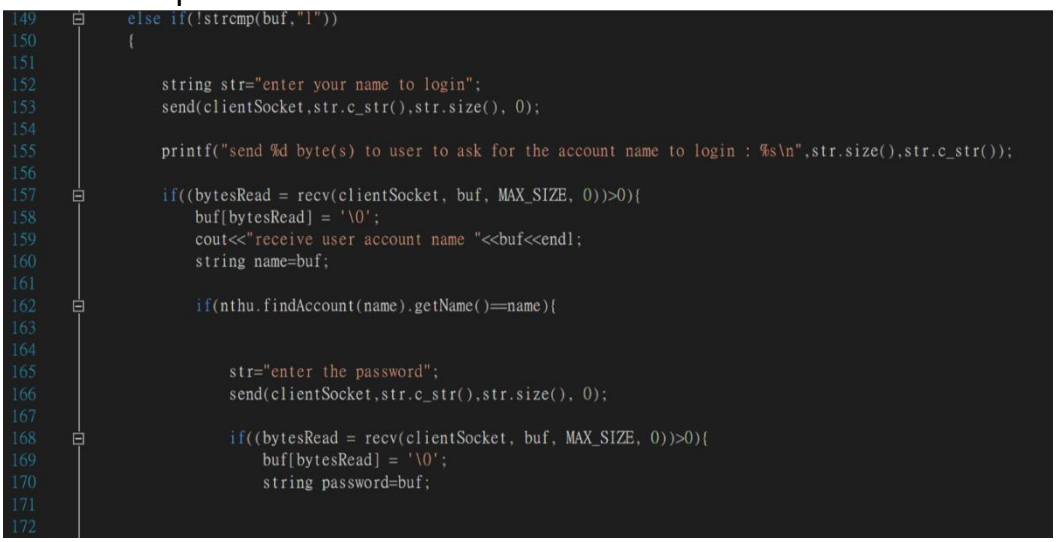
Next, whenever a client send commands to the server and the server receive them **via a while loop** , the system will parse the command into different instruction.

To be more specific, they support several commands.

1. Welcome menu:
    1. Login(l): login user account
    2. Create(c): create a user account
2. Command menu: After the user logins, it will shows the commands available for the user
    1. Deposit(d): allow the user to deposit money to the bank
    2. Withdraw(w):user could withdraw money from the bank if there is enough money in it.
    3. Check Info(i):display the user's information about his of her bank account.
    4. Exit(exit):permit the user to exit the bank system.

Here are the detailed implementations for each command

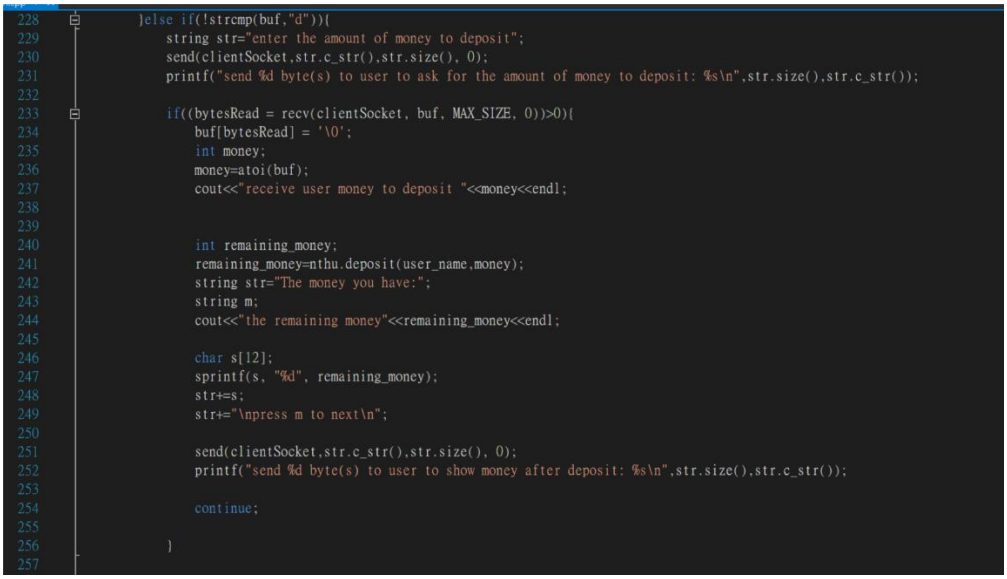
- Exit():receive “exit” from user
  - Whenever the server receive the “exit” message from the user, it invoke the nthu.update() to update the entire users’ information into BankData.txt which is the database.
  - Next, it send “Bye Bye” message back to the user to notify him.
  - In the end, the command instructions changes back to welcome message and the system then wait for another user to enter.
- Create():receive “c” from user
  - Upon receiving “c” from user, it asks for name user want to create.
  - If the account already exist, that is, nthu.findAccount(name).getName()==name, the creation will be denied.
  - Otherwise, it then asks for the password to type in and thus the creation will be success.
  - Notice that since it is echo server/client, after user complete each request, user need to press “m” before perform another instruction.



```
149     else if(!strcmp(buf,"1"))
150     {
151
152         string str="enter your name to login";
153         send(clientSocket,str.c_str(),str.size(), 0);
154
155         printf("send %d byte(s) to user to ask for the account name to login : %s\n",str.size(),str.c_str());
156
157         if((bytesRead = recv(clientSocket, buf, MAX_SIZE, 0))>0){
158             buf[bytesRead] = '\0';
159             cout<<"receive user account name "<<buf<<endl;
160             string name=buf;
161
162             if(nthu.findAccount(name).getName()==name){
163
164
165                 str="enter the password";
166                 send(clientSocket,str.c_str(),str.size(), 0);
167
168                 if((bytesRead = recv(clientSocket, buf, MAX_SIZE, 0))>0){
169                     buf[bytesRead] = '\0';
170                     string password=buf;
171
172
```

```
                    str="enter the password";
                    send(clientSocket,str.c_str(),str.size(), 0);
                    if((bytesRead = recv(clientSocket, buf, MAX_SIZE, 0))>0){
                        buf[bytesRead] = '\0';
                        string password=buf;
                        nthu.addCount(name,0,password);
                        str="The account is created\n";
                        str+="Press m to continue\n";
                        send(clientSocket,str.c_str(),str.size(), 0);
                        continue;

                    }
                }
            }
            else if(!strcmp(buf,"l")){
                if(nthu.authenticate(name,password)==true){
                    str="The authentication success\n";
                    str+="Your current money: ";
                    user_name=name;
                    int money=nthu.findAccount(user_name).getMoney();
                    char s[12];
                    sprintf(s, "%d",money);
                    str+=s;
                    str+="\n";
                    str+="Press m to continue\n";


                    send(clientSocket,str.c_str(),str.size(), 0);


                    command="\nDeposit(d)\n";
                    command+="Withdraw(w)\n";
                    command+="Check Info(i)\n";
                    command+="Exit(exit)\n";
                    continue;

                }else{

                    str="The authentication fail\n";
                    str+="Press m to continue\n";
                    send(clientSocket,str.c_str(),str.size(), 0);
                    continue;
                }
            }
        }
        else{
            string str="No such account\n";
            str+="Press m to continue to create a new account\n";
            send(clientSocket,str.c_str(),str.size(), 0);
            continue;

        }
    }
```

- Login(l): :receive “l” from the user
  - Allow user to login the system
  - First of all, user must to enter the name , and the server will check whether it is exist or not
  - Secondly, if it exist, it will check if the code matches the password via if(nthu.authenticate(name,password)==true)
  - Lastly, if the code user enter is correct, the authentication succeeds and the server display the current money in the user’s account by money=nthu.findAccount(user_name).getMoney();



```
228         }else if(!strcmp(buf,"d")){
229             string str="enter the amount of money to deposit";
230             send(clientSocket,str.c_str(),str.size(), 0);
231             printf("send %d byte(s) to user to ask for the amount of money to deposit: %s\n",str.size(),str.c_str());
232 
233             if((bytesRead = recv(clientSocket, buf, MAX_SIZE, 0))>0){
234                 buf[bytesRead] = '\0';
235                 int money;
236                 money=atoi(buf);
237                 cout<<"receive user money to deposit "<<money<<endl;
238 
239 
240                 int remaining_money;
241                 remaining_money=nthu.deposit(user_name,money);
242                 string str="The money you have:";
243                 string m;
244                 cout<<"the remaining money"<<remaining_money<<endl;
245 
246                 char s[12];
247                 sprintf(s, "%d", remaining_money);
248                 str+=s;
249                 str+="\npress m to next\n";
250 
251                 send(clientSocket,str.c_str(),str.size(), 0);
252                 printf("send %d byte(s) to user to show money after deposit: %s\n",str.size(),str.c_str());
253 
254                 continue;
255 
256             }
257 
```

- deposit (d): :receive “d” from the user
  - users can deposit money into their accounts.
  - Server receives the amount of money from the user(money=atoi(buf);) and deposit it to its account **remaining_money=nthu.deposit(user_name, money);**
  - After depositing the money to the user’s account, the server will show the remaining money.

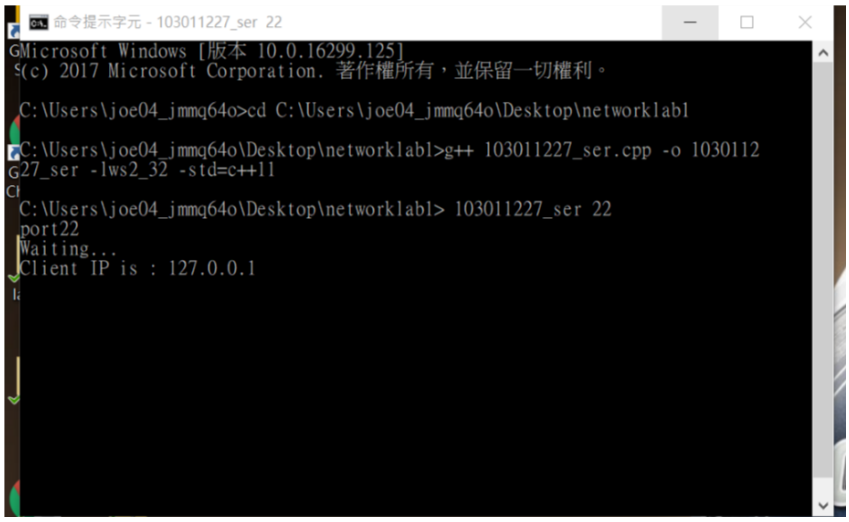
```
258             }else if(!strcmp(buf,"w")){
259                 string str="enter the amount of money to withdraw";
260                 send(clientSocket,str.c_str(),str.size(), 0);
261                 printf("send %d byte(s) to user to ask for the amount of money to withdraw: %s\n",str.size(),str.c_str());
262                 if((bytesRead = recv(clientSocket, buf, MAX_SIZE, 0))>0){
263                     buf[bytesRead] = '\0';
264                     int money=atoi(buf);
265                     cout<<"receive user money to withdraw "<<money<<endl;
266                     int remaining_money;
267                     remaining_money=nthu.withdraw(user_name,money);
268                     if(remaining_money>=0){
269                         string str="The money you have:";
270                         string m;
271                         char s[12];
272                         sprintf(s, "%d", remaining_money);
273                         str+=s;
274                         str+="\npress m to next\n";
275                         send(clientSocket,str.c_str(),str.size(), 0);
276                         printf("send %d byte(s) to user to show money after withdraw: %s\n",str.size(),str.c_str());
277                     }else{
278                         string str="you do not have enough money\n";
279                          str+="press m to next\n";
280                         send(clientSocket,str.c_str(),str.size(), 0);
281                         printf("send %d byte(s) to user to show money after withdraw: %s\n",str.size(),str.c_str());
282                     }
283                     continue;
284                 }
285
```

- withdraw (w): receive “w” from the tuser
  - Users could withdraw money from their bank account
  - Firstly, server receives the amount of the money user want to withdraw. That is to say, remaining_money=nthu.withdraw(user_name, money);
  - If there is enough money(if(remaining_money>=0)), the withdraw will succeed and the server will display remaining money.
  - Otherwise, the user does not have enough money and thus the server will deny this transaction

```
286         }else if(!strcmp(buf,"i")){
287             string name="Account Info: ";
288             name+=nthu.findAccount(user_name).getName();
289             int money=nthu.findAccount(user_name).getMoney();
290             char s[12];
291             sprintf(s, "%d",money);
292             name+=" ";
293             name+=s;
294             cout<<"Check account info"<<name<<endl;
295
296             name+="\nPress m to continue\n";
297             send(clientSocket,name.c_str(),name.size(), 0);
298
299             continue;
300
301         }else if(!strcmp(buf,"m")){
302             //send(clientSocket," ", command.size(), 0);
303            // continue;
304         }else{
305
306             string str="No such command\n";
307             str+="Press m to continue\n";
308             send(clientSocket,str.c_str(),str.size(),0);
309             continue;
310
311
312         }
313
```

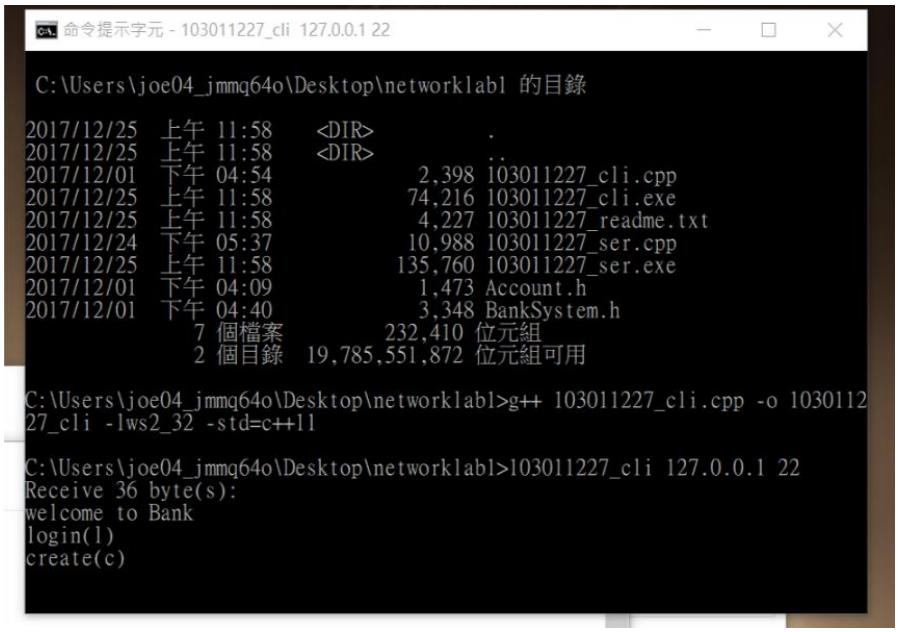
- info (i): :receive “i” from the user
  - server will display the user’s information about his name and money
  - nthu.findAccount(user_name).getName to get its name
  - money=nthu.findAccount(user_name).getMoney() to get its money
- m :receive “m” from the user
  - Because it is an echo server/client, pressing “m” allows the server to send menu message to the user and then user can perform next instruction.
- If user type any command that can not match any valid command, it will regarded as **invalid**.

4. Running Result (by screen)
    1. Compile the program and initiate the program
       Server: port=22
       Client: IP=127.0.0.1,port=22
        i. Server:



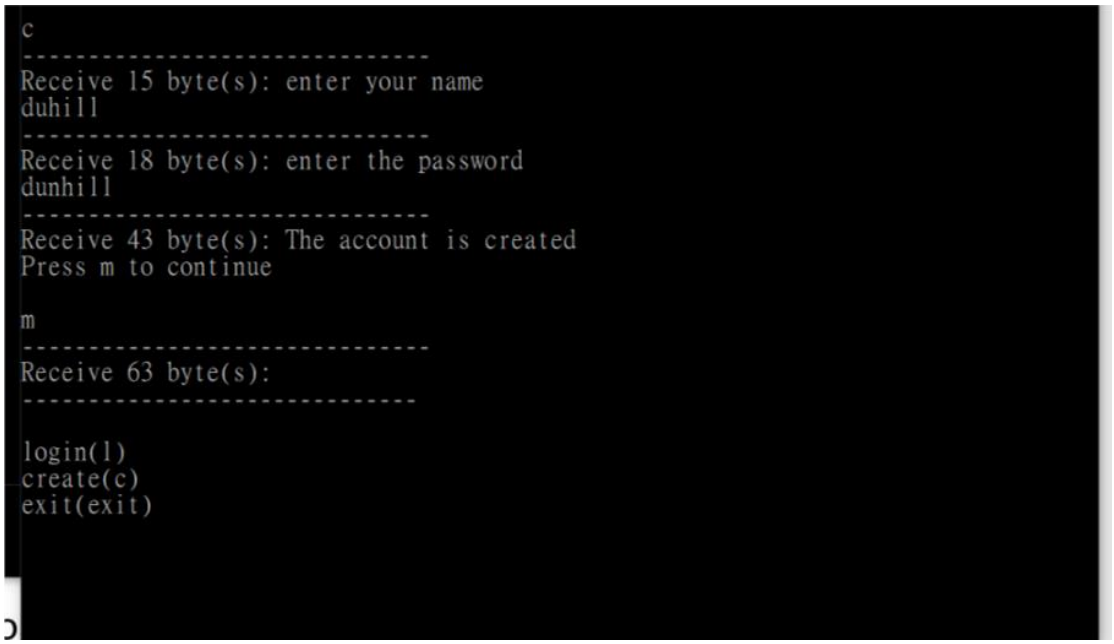
       Server will display the client's IP address after connecting with each other.

        ii. Client:



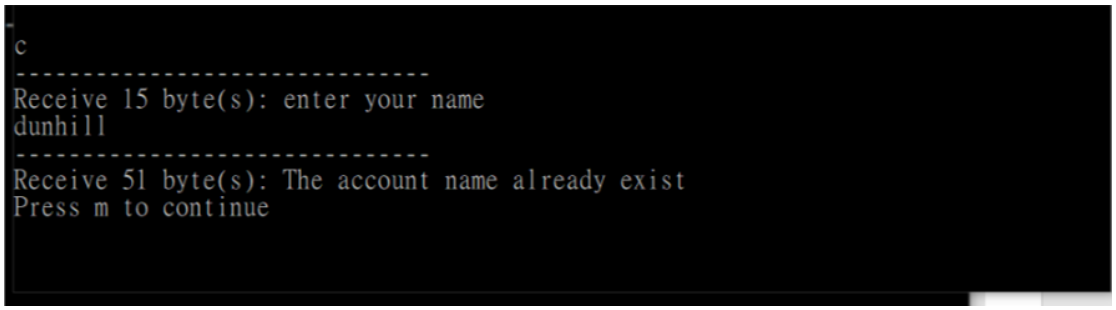
       Client will receive login and create menu.

    2. The user create a new account:
        i. If it is a new account:



The user type its name and password, and its money is set to 0 in the very first time.

ii. If the name already exists, the user cannot create an account.

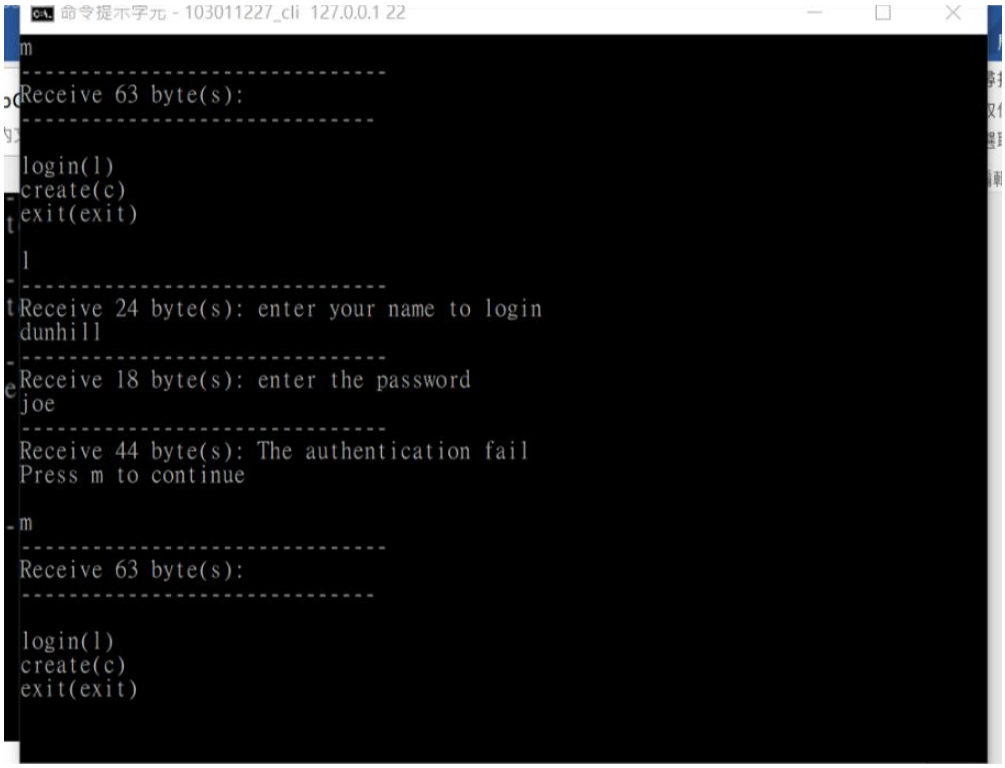


3. The user login a new account:

i. If the user name exist and the password is correct:

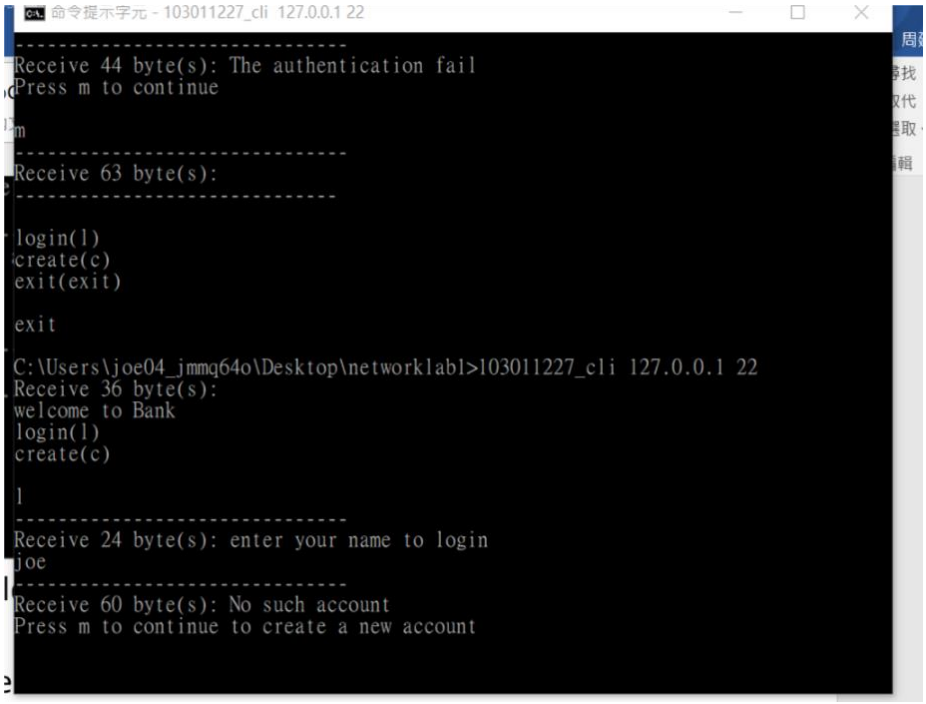
```
l
------------------------------
Receive 24 byte(s): enter your name to login
dunhill
------------------------------
Receive 18 byte(s): enter the password
dunhill
------------------------------
Receive 69 byte(s): The authentication success
Your current money: 0
Press m to continue

m
------------------------------
Receive 50 byte(s):
Deposite(d)
Withdraw(w)
Check Info(i)
Exit(exit)
```

ii. If the user enter the wrong password:

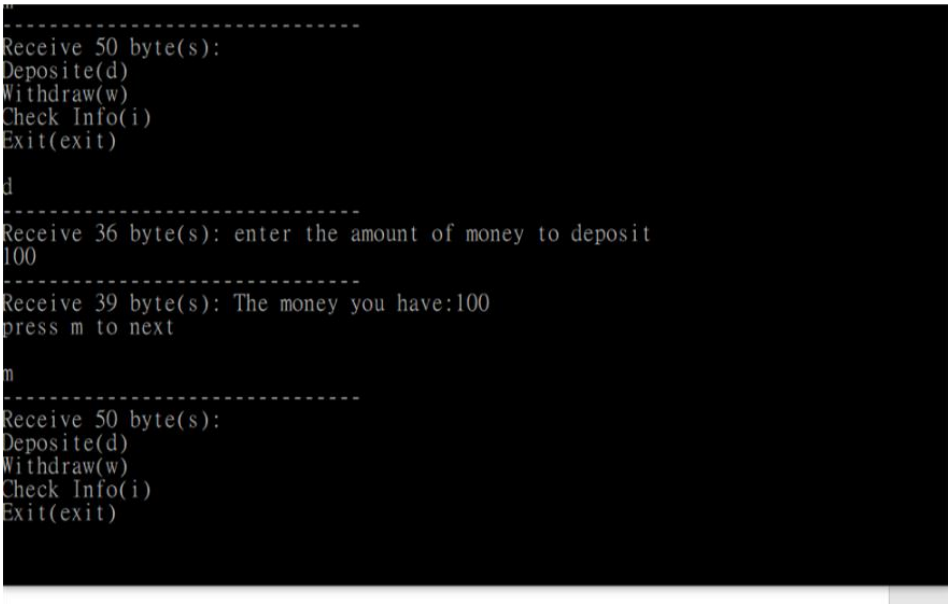


The password should “dunhill” but the user type “joe” instead. As a result, the authentication fails.

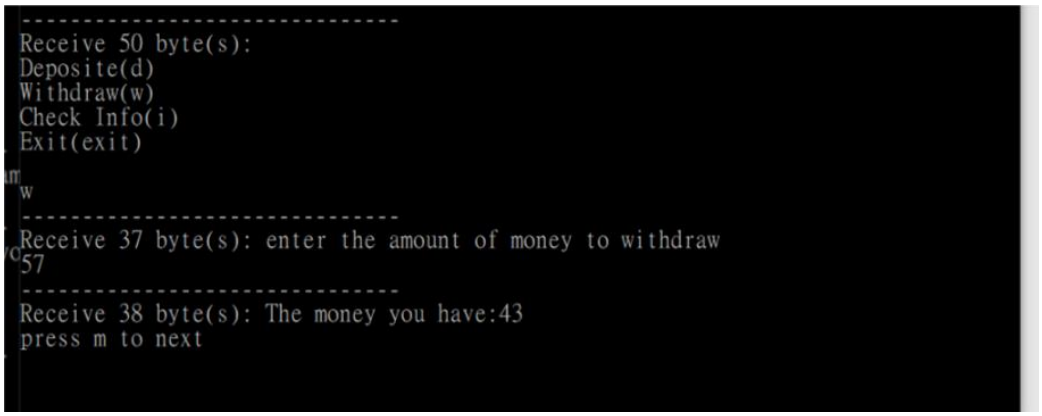
iii. If the account name does not exist.



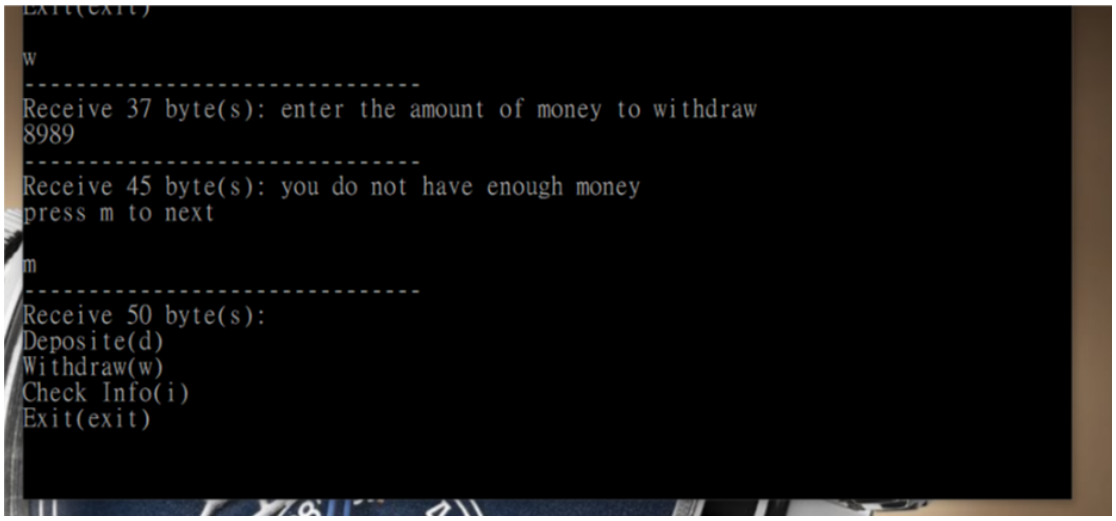
4. The user can deposit money to his/her account



5. The user can withdraw money
   i. If there exists enough money

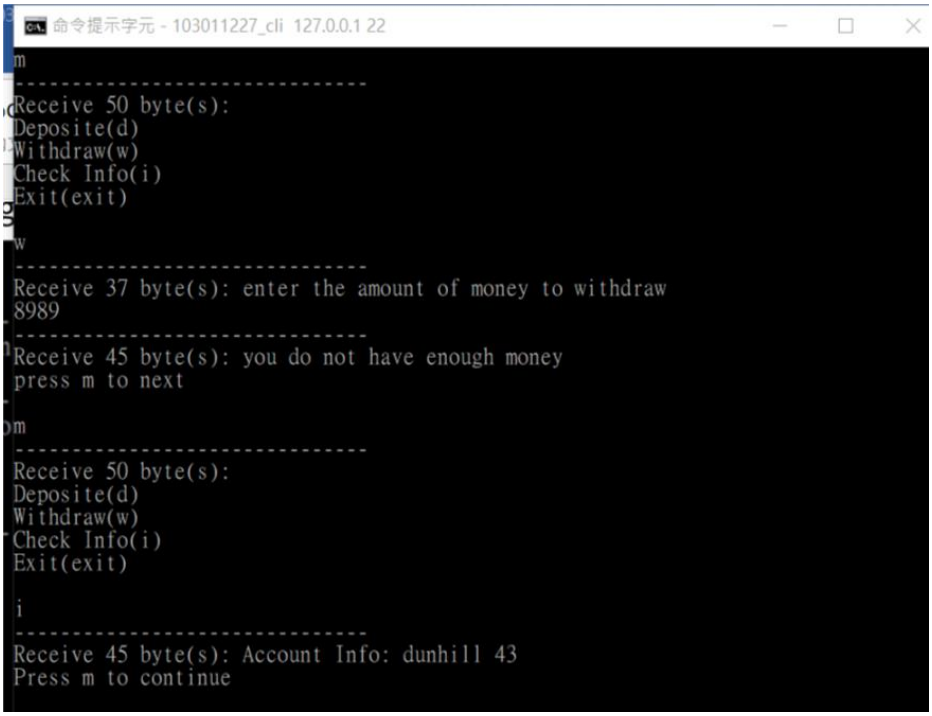


ii. If there is not enough money:



Consequently, the transaction will deny.

6. The user can check his/her account information:



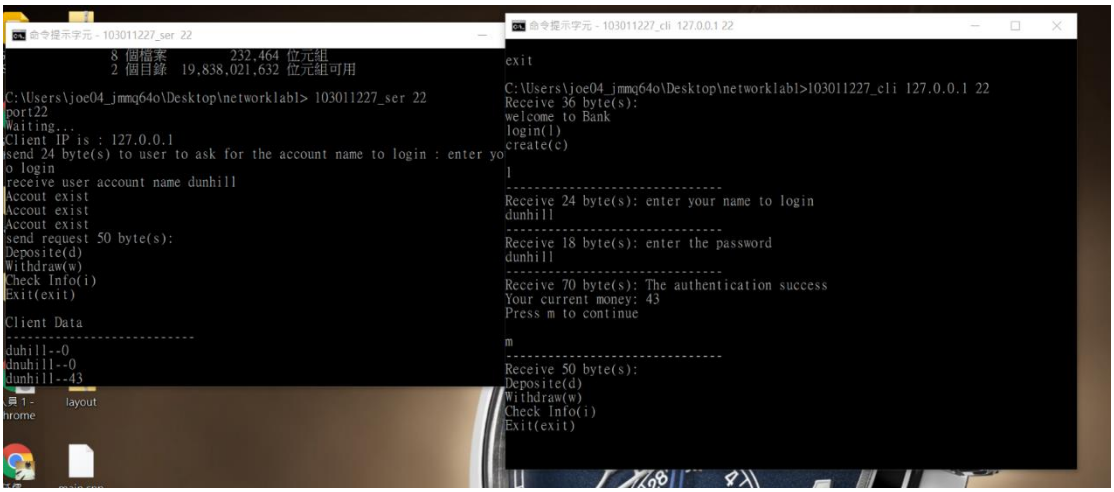
The server will display user's name and money.

7. The user can leave the system

8. If the user type invalid command:

The user type "invalid" which is not valid.

9. Before the server shut down, it will save the all account data into the database. Thus, it will restore the data from the database once it restarts.



5. Reviews:
   In this lab, we are required to implement TCP/IP transaction system. Since I have experience about socket programming from Network Programming course, learning winsock would require little effort. However, there still exist some difficulties in this lab.

   First of all, because it is designed based on the echo server/client, the modifications to the architecture would be quite difficult.
   To be specific, a user cannot send more than 2 messages to the server at once. If a user want to send more than 2 messages, he/she are required to the first message, wait for response from the server, and then send the second message. To solve the problem, we add a non-function command “m” to server. And then the user/client can thus send second message.

   Secondly, the database plays an important role in the transaction system. That is to say, database can store the information data even if the system is turned off or shut down. Strictly speaking, it should be implemented via SQL. But for simplicity, we implement the database via a txt file. And in the txt file, it records the account’s name, password and the money respectively. Whenever a user leaves the system, the system update the data information in the txt file. In the end, the system can load the entire user files from the txt file whenever the system initiated

   To sum up, in the lab, we enhance our skills about socket programming and grasp comprehensive understanding of network. After all, socket API plays a crucial role in TCP/IP, and socket programming skill would be indispensable.