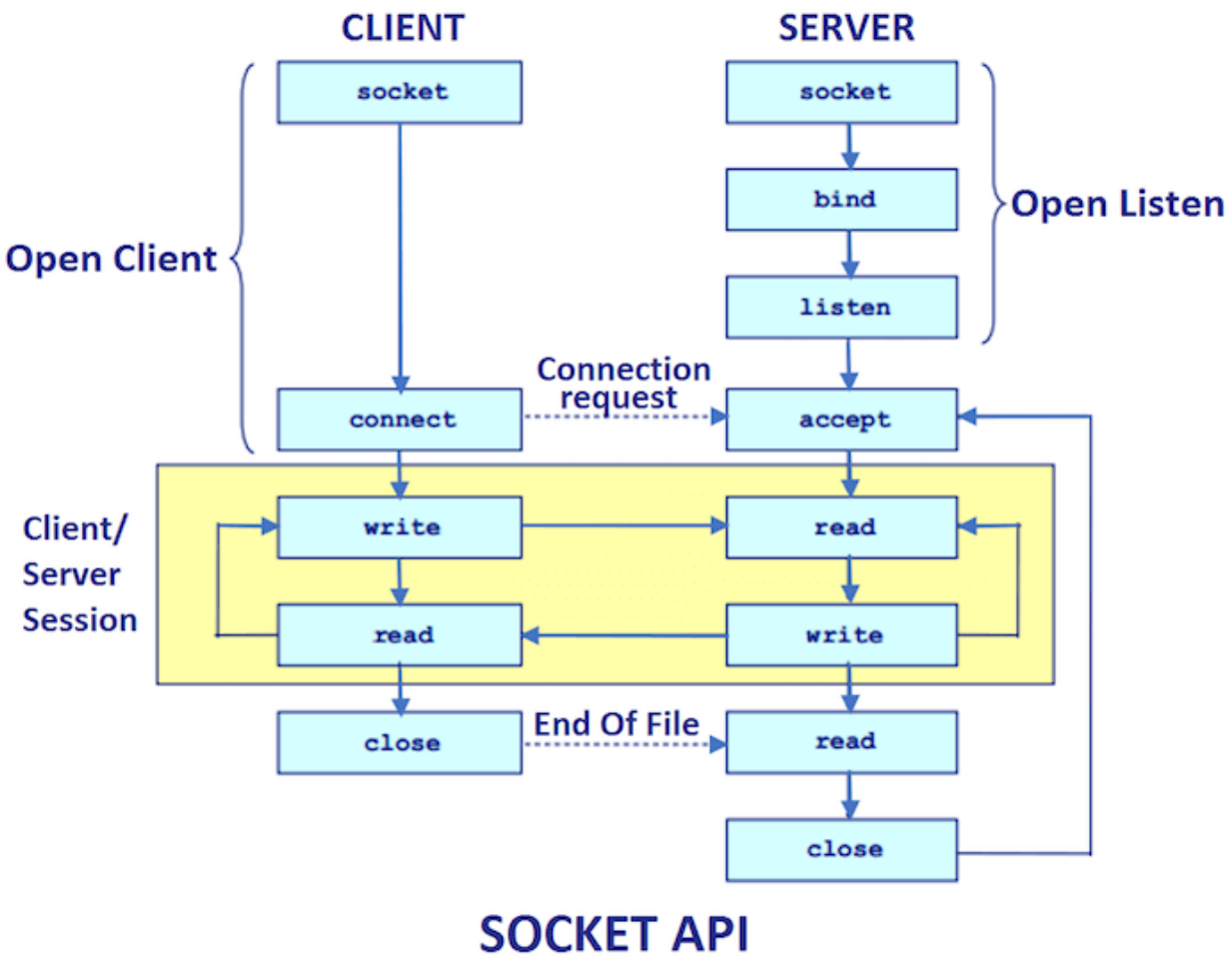
Susanka Majumder
001810501053
IT Assignment - 1 (Implement a TCP-based key-value store)

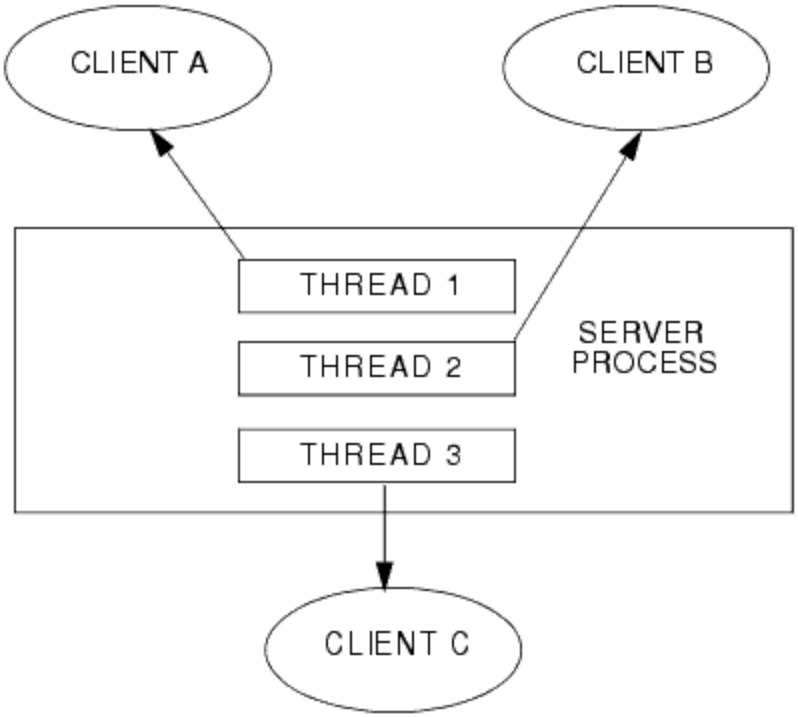
Problem Statement : Implement a TCP-based key-value store. The server implements the key-value store and clients make use
of it. The server must accept clients' connections and serve their requests for 'get' and 'put' key value pairs.
All key-value pairs should be stored by the server only in memory. Keys and values are strings.
The client accepts a variable no of command line arguments where the first argument is the server hostname
followed by port no. It should be followed by any sequence of "get <key>" and/or "put <key> <value>".
./client 192.168.124.5 5555 put city Kolkata put country India get country get city get Institute
India
Kolkata
<blank>
The server should be running on a TCP port. The server should support multiple clients and maintain their
key-value stores separately.
Implement authorization so that only few clients having the role "manager" can access other's key-value
stores. A user is assigned the "guest" role by default. The server can upgrade a "guest" user to a "manager"
User.

Design :

This project is based on server client architecture which is implemented using sockets in C++ .
The basic socket API design is described below .



When a socket connection is accepted a thread is created in the server dedicated to that (client) connection and then the main thread listens to other upcoming connections this way we can handle multiple clients without worrying about concurrent access .



IMPLEMENTATION : -

Server.cpp - consists of the server functionalities .

Socket creation and binding to a specific port .

```
 sockaddr_in serv, client; //main server variable.
   int fd; //Socket file descriptor that will be used to identify the
socket
   int conn; //Connection file descriptor that will be used to distinguish
client connections.

   char* host_addr = "127.0.0.1" ;

   int PORT = 8888 ;

   if(argc>1)
       PORT = atoi(argv[1]) ;
```

```
   if(argc>2)
       host_addr = argv[2] ;


   serv.sin_family = AF_INET;
   serv.sin_port = htons(PORT); //Define the port at which the server will
listen for connections.
   serv.sin_addr.s_addr = inet_addr(host_addr); // or use INADDR_ANY

   if((fd = socket(AF_INET, SOCK_STREAM,0) )<0){ //AF_UNIX
       cerr<<"No sock Descriptor"<<endl;
       return -1;
   }

   int opt = 1;
   if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                  &opt, sizeof(opt)))
   {
       cerr<<"setsockopt";
       return -1;
   }

   if (bind(fd, (struct sockaddr *) &serv, sizeof(serv))< 0){
       cerr<<"Bind error";
       return -1;
   } //assigns the address specified by serv to the socket
```

Listen to incoming connections .

```
 if(listen(fd, 1)<0){
       cout<<"Unable to Listen";
       return -1;
   } //Listen for client connections

   cout<<"Server Running on IP: "<<inet_ntoa(serv.sin_addr)<<" PORT:
"<<ntohs(serv.sin_port)<<endl;
```

Handling connection with multi threading . a thread to created every time a new socket connection is established and the rest of the server functionalities are executed in that thread

```
//Now we start handling the connections.
   int addlen = sizeof(client);

    while (conn = accept(fd, (struct sockaddr *) &client, (socklen_t
*)&addlen)){
       int pid;
       cout<<"New Connection with Client IP:
"<<inet_ntoa(client.sin_addr)<<" Client PORT:
"<<ntohs(client.sin_port)<<endl;
       thread t(main_program,conn) ;
       t.detach();
   }
```

Handles the response to authentication prompt .

```
void prompt_acceptance(int& sock , string& authentication_string){
   char prompt_buffer[1024] ;
   recv(sock , prompt_buffer , 1024 , 0 ) ;
   string prompt_string(prompt_buffer) ;


   if(prompt_string[0] == 'y')
       authenticate_manager(sock,authentication_string) ;
}
```

Authenticate manager w.r.t the username and passwords that are already stored in server

```
const string predefined_username = "manager" ;
const string  predefined_password = "123456" ;

 const int BUFFERSIZE = 2048;
```

```
void authenticate_manager(int& new_socket, string& authentication_string
){


                char username_buffer[1024] , password_buffer[1024] ;

                string username_string , password_string  ;

                read(new_socket , username_buffer , 1024 ) ;

                username_string = username_buffer ;

                read(new_socket , password_buffer , 1024 ) ;


                password_string = password_buffer ;


                authentication_string =
(username_string==predefined_username && password_string ==
predefined_password)?"authenticated":"unatuhenticated" ;

                if( send(new_socket ,authentication_string.data() ,
authentication_string.size() , 0 ) != authentication_string.size() ){
                    perror("authentication_string sending failed");
                }


            }
```

Continuously listen for any query are sent from the shell . read the incoming buffer and decode the query string .

```
while (recv(conn, query_string, BUFFERSIZE, 0) > 0) {


                int counter = 0 ;
```

```
                    // cout<<query_string<<endl ;

                     for(int i = 0 ; i < strlen(query_string)  ; i++ ){

                      //   cout<<counter<<endl ;

                         if(query_string[i] =='#')
                         {
                             counter++ ;
                         }
                         else
                         {
                             query_info[counter] += query_string[i] ;
                         }
                     }

                         printf("received query : ");
                         cout<<query_info[0]<<" "<<query_info[1]<<"
"<<query_info[2]<<" "<<query_info[3]<<" "<<query_info[4]<<endl ;

                         if(query_info[0] == "put"){
                         key = query_info[1] ;
                         value = query_info[2] ;
                         int id = atoi( query_info[3].c_str() );
                         //cout<<key<<"          "<<value<<"\n" ;
                         response_string = put_value_guest(key , value , id
) ;
                     }
                     else if(query_info[0] == "get" ){
                             key = query_info[1] ;
                         //value = query_info[2] ;
                         int id = atoi( query_info[3].c_str() );
                         response_string = get_value_guest(key , id ) ;
                     }
                     else if(query_info[0] == "putm"){
                         key = query_info[1] ;
                         value = query_info[2] ;
                         int id = atoi( query_info[3].c_str() );
                         int cli_id =  atoi( query_info[4].c_str() );
```

```
                        response_string = put_value_manager(key , value ,
cli_id , authentication_string ) ;
                    }
                    else if(query_info[0] == "getm" ){
                            key = query_info[1] ;
                        //value = query_info[2] ;
                        int id = atoi( query_info[3].c_str() );

                         int cli_id =  atoi( query_info[4].c_str() );

                            response_string = get_value_manager(key ,
cli_id , authentication_string ) ;

                    }
                    else if(query_info[0] != "exit" ){
                        response_string = "invalid query :( , try again ."
;
                    }


                    send(conn, response_string.data() ,
response_string.size() , 0 ) ;


                    query_info.clear() ;
                    query_info.assign(10 , "") ;
                    response_string ="" ;
                    bzero(query_string,BUFFERSIZE) ;
                    memset(query_string, 0, sizeof(query_string));
            }
```

Invoke particular function according to the query type .

The database is saved as a vector of map with <string , string> key value pair .

```
vector<map<string , string> > global_database(1000) ;
```

Different query functions that have been implemented are .

```
string get_value_guest(string key , int id){

   if(global_database[id][key] != "")
       return global_database[id][key] ;
   return ("no entry found :(");
}

string put_value_guest(string key , string value , int id){
    global_database[id][key] = value ;
   return ("added to database (" + key + "," + value + ")"  );
}

string put_value_manager(string key , string value , int id,string
authentication_string){
   if(authentication_string != "authenticated" )
       return "you are not logged in as a manager UwU" ;
   global_database[id][key] = value ;
   return ("added to database (" + key + "," + value + ") of client " +
to_string(id) );
}

string get_value_manager(string key , int id,string
authentication_string){

    if(authentication_string != "authenticated" )
       return "you are not logged in as a manager UwU" ;

   if(global_database[id][key] != "")
       return global_database[id][key] ;
   return ("no entry found :(");
}
```

That’s pretty much it on the server part .

Client.cpp : handles all the client functionalities .

```
int PORT = 8888 ;

   if(argc> 2 )
       PORT = atoi(argv[2]) ;

   string client_id ;

   if(argc>1)
       client_id = argv[1] ;

   char* serv_address = "127.0.0.1" ;
   if(argc > 3)
       serv_address = argv[3] ;
   int sock = 0, valread;
   struct sockaddr_in serv_addr;
   string hello = "Hello from client";
   char buffer[1024] = {0};
   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
       printf("\n Socket creation error \n");
       return -1;
   }

   serv_addr.sin_family = AF_INET;
   serv_addr.sin_port = htons(PORT);

   // Convert IPv4 and IPv6 addresses from text to binary form
   if(inet_pton(AF_INET, serv_address, &serv_addr.sin_addr)<=0)
   {
       printf("\nInvalid address/ Address not supported \n");
       return -1;
   }

   if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) <
0)
   {
       printf("\nConnection Failed \n");
       return -1;
   }
```

Socket Connection Established .

```
   void authentication_prompt(int& sock){

   printf("welcome aboard\n would you like to get authenticated as a
manager ? [y/n] " );
   string prompt_response ;
   cin>>prompt_response ;
   sendsync(sock , prompt_response) ;
   if(prompt_response=="y")
       autheticate_manager(sock) ;
}
```

Authentication prompt asks the user whether he wants to get authenticated as a manager once the client gets initiated .

```
bool autheticate_manager(int& sock ){
           string username , password ;
           cout<<"username: " ;
           cin>>username ;
           cout<<endl ;
           cout<<"password: " ;
           cin>>password ;
           cout<<endl ;
           send(sock , username.data() , username.size() , 0 ) ;
           sleep(2) ;

           send(sock , password.data() , password.size() , 0 ) ;
           //sleep(2) ;

           char authentication_buffer[1024]  = {0}  ;

           recv( sock , authentication_buffer , 1024 , 0 ) ;
           //sleep(2);
```

```
            string authentication_string  = authentication_buffer ;

            //printf("%s\n", authentication_string );

            cout<<authentication_string<<endl ;

            if(authentication_string == "authenticated" )
            {
                mode = "manager" ;
                printf( COLOR_GREEN "manager authentication successful
\n\n") ;
                return true ;

            }
            else if( authentication_string == "unauthenticated" ){
                printf( COLOR_RED "wrong username of password \n\n") ;
            }
            else{
                cout<<"having issue while authenticating \n";
            }
            return false ;

    }
```

This servers the purpose of authenticating the manages

```
while(true){
        printf( COLOR_BLUE "%s-%s@" COLOR_CYAN "query-shell> " COLOR_NONE ,
mode , argv[1]) ;
        cin>>query_string ;
        string query_info ;
        if(query_string =="put")
        {
            cin>>key>>value ;
            query_info = query_string + "#" + key + "#" + value + "#" +
client_id + "#" ;
```

```
        }
        else if(query_string == "get")
        {
            cin>>key ;
            query_info = query_string + "#" + key + "#" + value + "#" +
client_id + "#" ;
        }
        else if(query_string == "putm")
        {
            cin>>key>>value>>_id ;
            query_info = query_string + "#" + key + "#" + value + "#" +
client_id + "#" + to_string( _id) + "#" ;
        }
        else if(query_string == "getm")
        {
            cin>>key>>_id ;
            query_info = query_string + "#" + key + "#" + value + "#" +
client_id + "#" + to_string( _id) + "#" ;
        }




        send(sock , query_info.c_str() , query_info.size()  , 0) ;


        if(query_string == "exit")
        {
            break ;

        }

        bzero(response_buffer , 1024 );

        recv(sock , response_buffer , 1024 , 0 ) ;

        printf( COLOR_YELLOW "%s\n", response_buffer );
        query_string = "" ; key ="" ; value ="" ; query_info="" ;
    }
```

Once the client is authenticated as manager or guest it enters an interactive query shell and gets the query input, encodes them and sends them to the servers and waits for response . once it gets the response back from the server it displays relevant information in the terminal .

OUTPUT : The screenshot below suggest how different authorization schemas and the commands work e.g (e.g manager having access to guests data but not the vice versa) .)