Assignment-13
ELP- 718 Telecom Software Laboratory

Anshuman Singh
2018JTM2004
2018-19

A report presented for the assignment on
Socket Programming in C

Bharti School
of
Telecommunication Technology and Management
IIT DELHI, Delhi
Novembar 12, 2018

Contents

List of Figures

1 Problem Statement-1

1.1 Problem Statement

Youve to design a simple client and server TCP communication using sockets.

1. Server before establishing connection with the client should ask client for credentials.If valid (authenticate user_id and password entered by the client) then allow connection otherwise prompt client again for id and password.
2. After successful authentication, the client sends a string to server and the server change the case of each alphabet and send it back to client.
3. The client take user input from console as to what string has to be sent.The returned string is displayed underneath.
4. The communication continues on as long as server or client dont close their sockets.
5. The different stages of TCP communication should be shown with corresponding messages as well. You should also display error in case of failures.

1.2 Assumptions

- Credentials are stored in a file named user_pass.
- All processing is done at the server.

1.3 Algorithm and Implementation [1] [2] [3]

Server Side:

- Create a socket file descriptor first.
- Specify the address family as IPv4 and IP address as the IP of the system.
- Bind the socket to a port number.
- Listen for connections from the clients.
- Now accept connections from clients and store their socket descriptors.
- Ask for credentials from the client.
- Authenticate or ask again for credentials.
- Create a child process using fork for each new connection.
- Create a function to receive the input string from the user.
- Then call the function written to process the query.

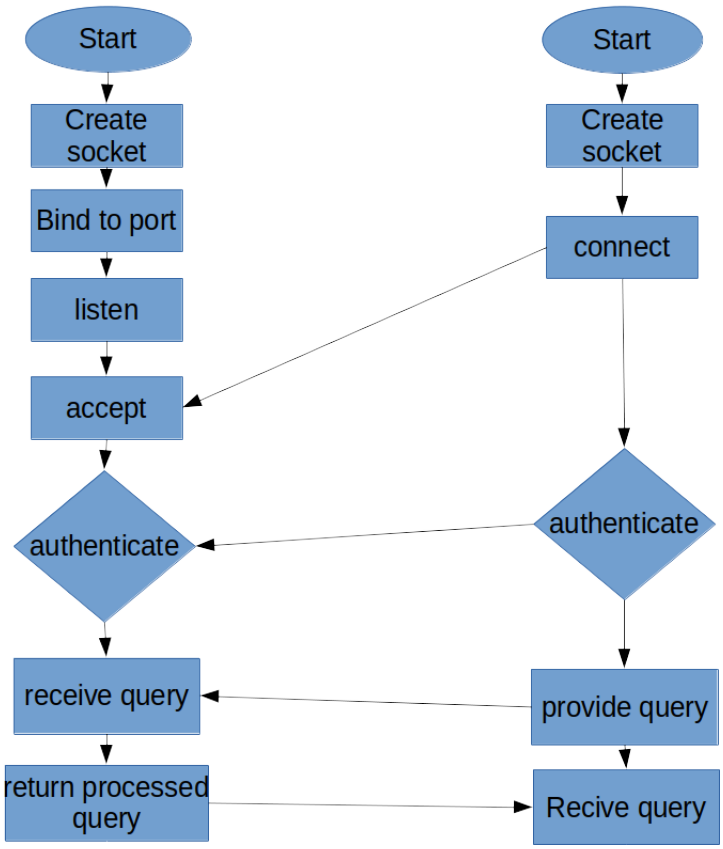
- Return the query and ask again for the next query.

Client Side:

- Create a socket file descriptor first.
- Specify the address family as IPv4 and IP address as the IP of the system.
- Bind to the port number on which server is running.
- Connect to server using the port number and address family specified.
- Provide credentials.
- Keep asking for query from user and process it.
- Re-enter the query for processing until quit is pressed.

1.4 Flow Chart

Figure 1: Flow Chart for Figure 1



1.5 Input and Output Format

- **Client Interface example(on connection)**:
  Please Enter Username: xyz
  Please Enter Password: 4567
  Enter the string: HelLo (if credentials are valid)
  Reply from server : hELlO
  Enter the string :

- **Server Interface example**:
  Validate the credentials
  String received : HelLo
  Replied string : hELlO

1.6 Screenshots

Figure 2: Terminal Output of Server Assignment 1

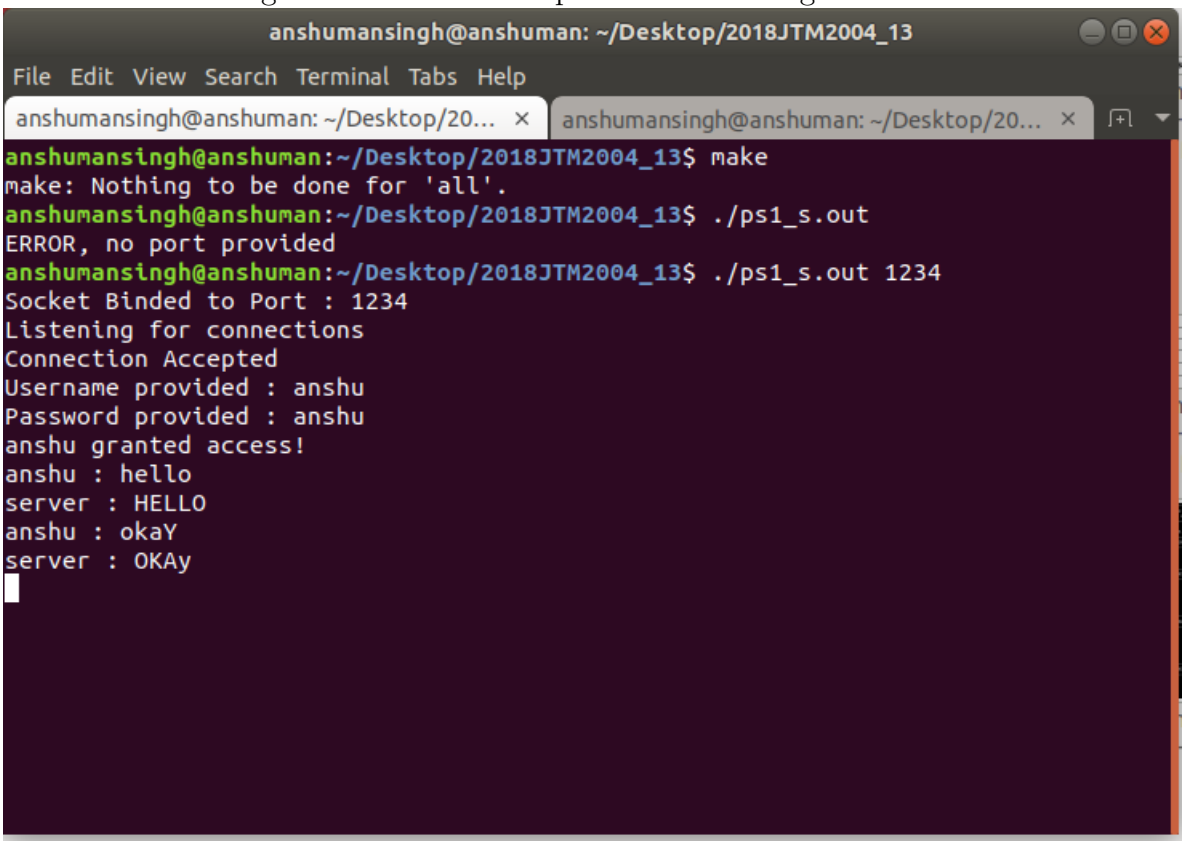
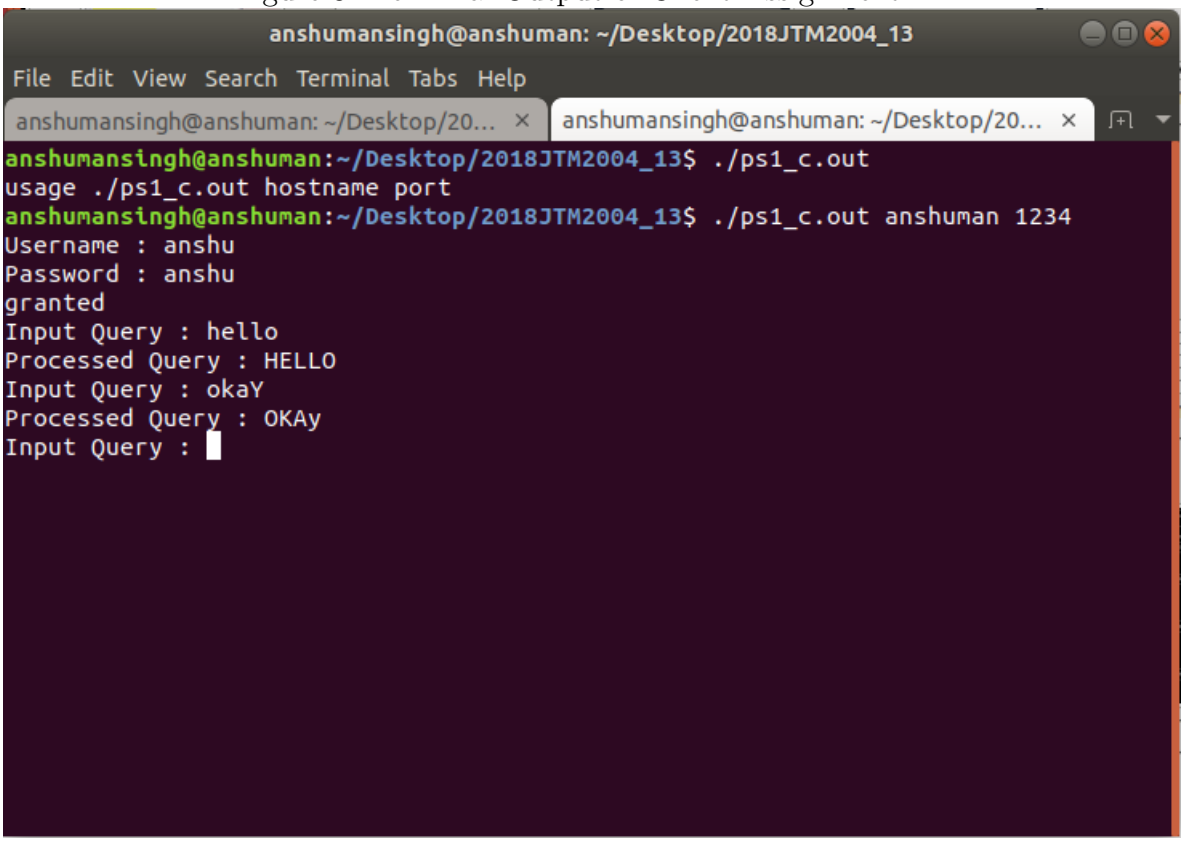


Figure 3: Terminal Output of Client Assignment 1



2 Problem Statement-2

2.1 Problem Statement

You have to create a server capable of handling multiple clients up to 5 and rejecting more than 5 connections using TCP communication sockets.Server should work in following phases.

1. Server before establishing connection with the client should ask client for credentials.If valid (authenticate user_id and password entered by the client) then allow connection otherwise prompt client again for id and password.

2. After successful authentication ,conversation should start between clients and server.Store the chat history between server and clients in a file chat.txt along with the timestamp (containing date and time)adjacent to the chat string.

3. One message send by a client should be broadcasted to all clients.

4. Before broadcasting the message server modify the message send by client by adding prefix to the message.

2.2 Assumptions

- Credentials are stored in a file named user_pass.
- Chat history gets stored in a file named chat_history.

2.3 Algorithm and Implementation [1] [2] [3]

Server Side:

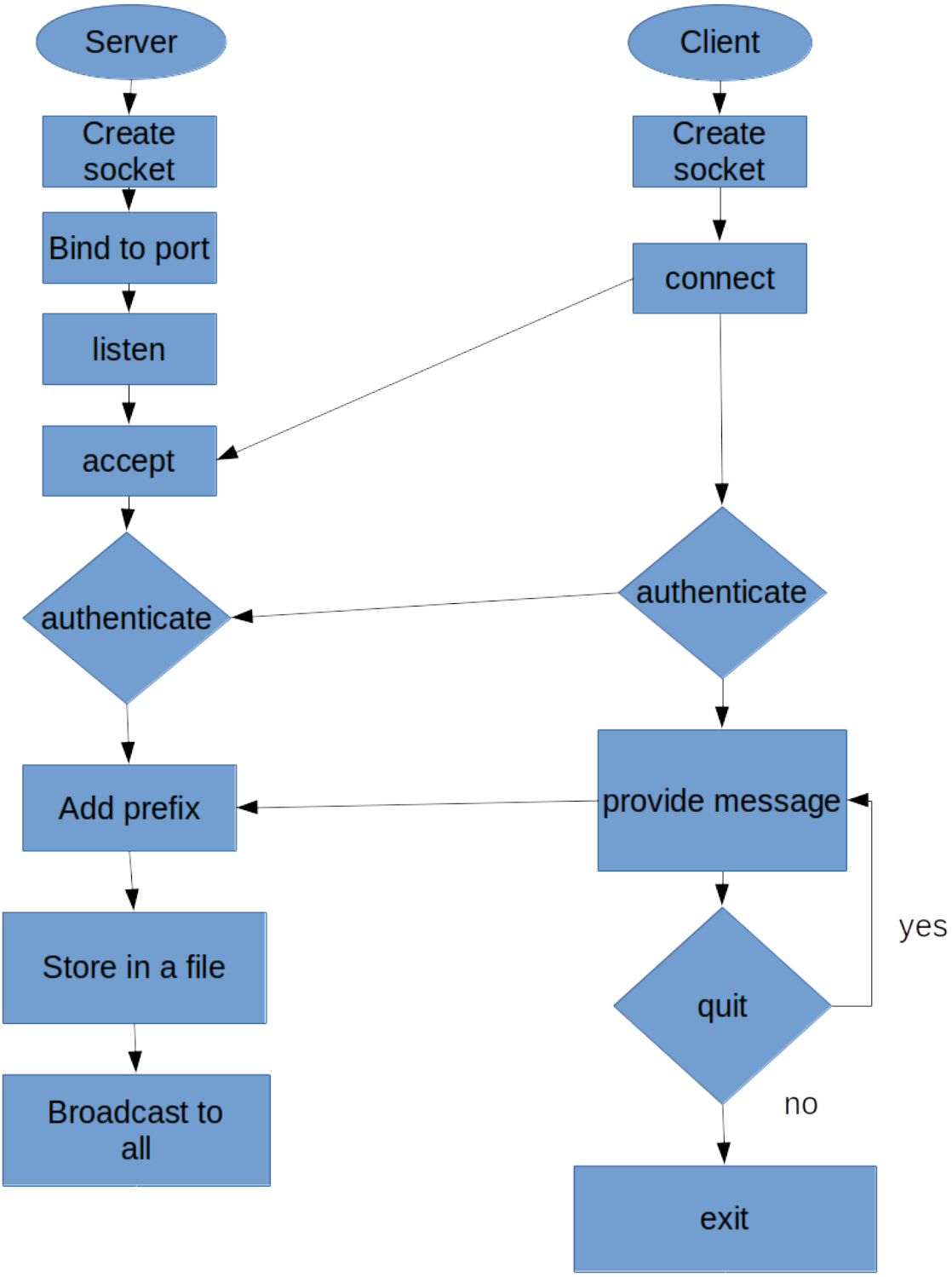
- Create a socket file descriptor first.
- Specify the address family as IPv4 and IP address as the IP of the system.
- Bind the socket to a port number.
- Listen for connections from the clients.
- Now accept connections from clients and store their socket descriptors.
- Ask for credentials from the client.
- Authenticate or ask again for credentials.
- Keep checking for data from all clients.
- When data is present broadcast it to all the clients along with prefix.
- Store the chat history in a file.
- When quit is given as input quit the client program.

Client Side:

- Create a socket file descriptor first.
- Specify the address family as IPv4 and IP address as the IP of the system.
- Bind to the port number on which server is running.
- Connect to server using the port number and address family specified.
- Provide credentials.
- Enter message to broadcast until quit is given.

2.4 Flow Chart

Figure 4: Terminal Output of Server Assignment 2



2.5 Input and Output Format

Client interface example:

- Please Enter Username: abc
- Please Enter Password: 1234
- Received String: Valid/Enter Credentials Again

Server Interface example(on connection):

- ./server.out xyz_
- xyz_ - prefix

2.6 Screenshots

Figure 5: Terminal Output of Server Assignment 2

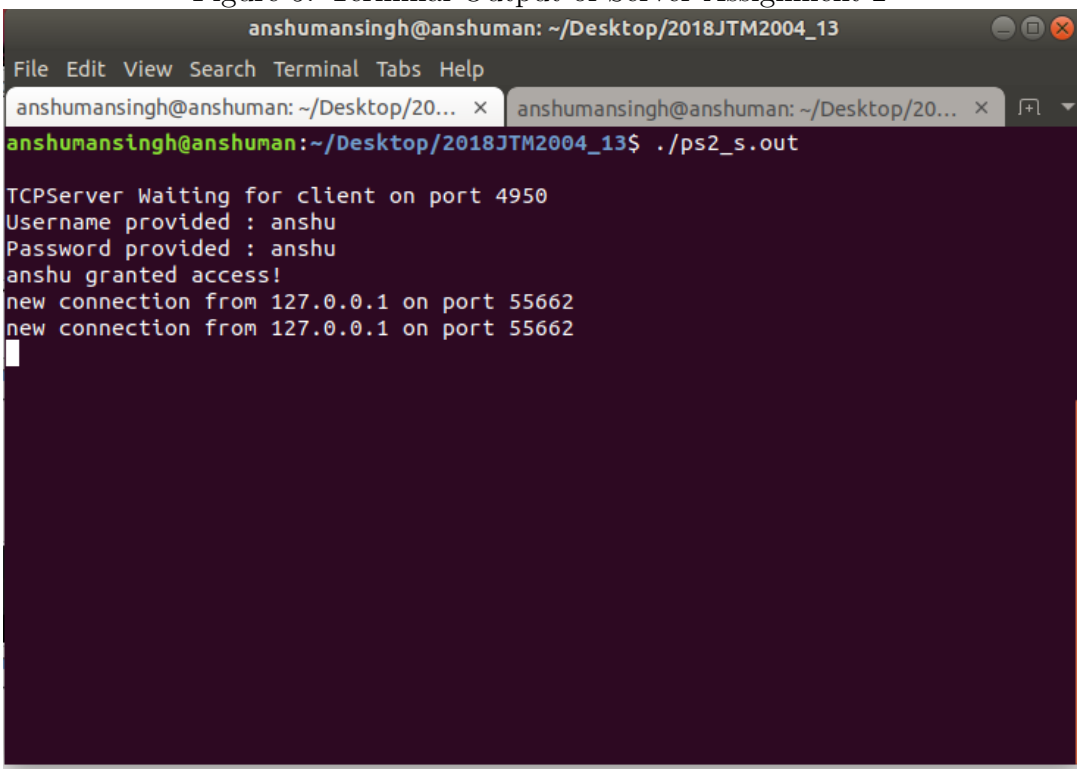
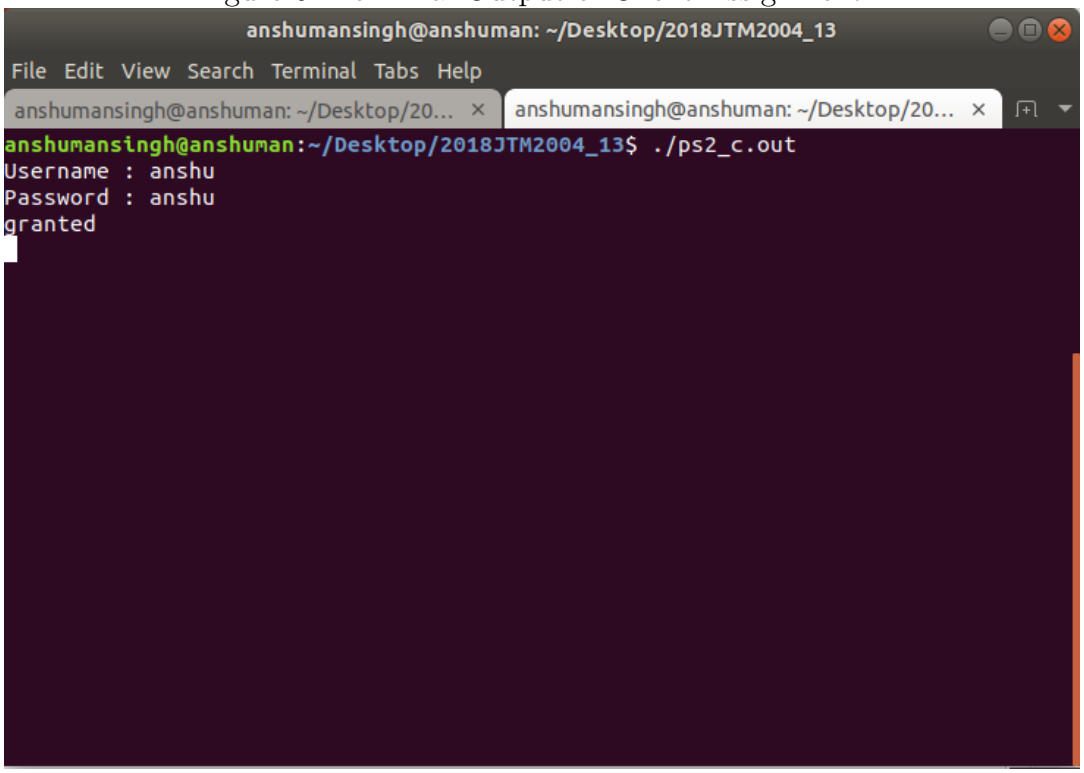


Figure 6: Terminal Output of Client Assignment 2



3 Appendix

3.1 Code for ps1_s

```
/* To run the server program
* 1. make -B
* 2. Example:  ./server.out 12345
*/

/*
* The program assumes a file nammed
* user_pass exists with username and
* password arranged in two columns.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h> //for read and write function

void attend(int);
void results(char *query);
int readsock(int sock,char buffer[]);
```

```
int writesock(int sock, char buffer[]);

void error(char *msg)
{
    perror(msg);
    exit(1);
}

int readsock(int sock,char buffer[]){
  int n;
  bzero(buffer,256);
  n = read(sock,buffer,256);
  if (n < 0) error("ERROR reading from socket");
  return n;
}

int writesock(int sock, char buffer[]){
  int n;
  n = write(sock,buffer,strlen(buffer));
  if (n < 0) error("ERROR writing to socket");
  return n;
}

int main(int argc, char *argv[])
{
     int sockfd, newsockfd, portno, clilen, pid;
     struct sockaddr_in serv_addr, cli_addr;

     if (argc < 2) {
         fprintf(stderr,"ERROR, no port provided\n");
         exit(1);
     }
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
     if (sockfd < 0)
        error("ERROR opening socket");
     bzero((char *) &serv_addr, sizeof(serv_addr));
     portno = atoi(argv[1]);
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     if (bind(sockfd, (struct sockaddr *) &serv_addr,
              sizeof(serv_addr)) < 0)
              error("ERROR on binding");
    else{
      printf("Socket Binded to Port : %d\n", portno);
    }
     listen(sockfd,2);
     printf("Listening for connections\n");
     clilen = sizeof(cli_addr);
     while (1) {

         // Accept new connection.
         newsockfd = accept(sockfd,(struct sockaddr *) &cli_addr, &
             clilen);
```

```
            if (newsockfd < 0) error("ERROR on accept");
            pid = fork();
            if (pid < 0) error("ERROR on fork");
            if (pid == 0)  {
                printf("Connection Accepted\n");
                close(sockfd);
                attend(newsockfd);
                exit(0);
            }
            else close(newsockfd);
        }
    return 0;
}
void results(char s[]){

//This function changes case of each character.
  int c = 0;
  char ch;

  while (s[c] != '\0') {
     ch = s[c];
     if (ch >= 'A' && ch <= 'Z')
        s[c] = s[c] + 32;
     else if (ch >= 'a' && ch <= 'z')
        s[c] = s[c] - 32;
     c++;
  }

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


}
void attend (int sock)
{
   int n;
   //Initialize array to hold username, password and other data.
   char buffer[256],username[256],password[256];
   char name[10],pass[10];

   auth:
   //Read username from socket
   n = readsock(sock,username); username[n-1] = '\0';
   printf("Username provided : %s\n",username);

   //Read password from socket
   n = readsock(sock,password); password[n-1] = '\0';
   printf("Password provided : %s\n",password);


   //Reading username and password from file user_pass
   FILE * fp = fopen("user_pass","r"); fscanf(fp, "%s %s",name,pass);
   // printf("%s %s\n",name,pass);

   // Compare username and password with Credentials from file.
```

```
    if (strcmp(username,name) == 0 && strcmp(password,pass) == 0 ){

      n = write(sock,"granted",strlen("granted")); if (n < 0) error("
          ERROR writing to socket");
      printf("%s granted access!\n",username);

      while(1){
        readsock(sock,buffer); // Reading Input string from Client
        printf("%s : %s\n",username, buffer );
        results(buffer); //process the string
        printf("server : %s\n",buffer );
        writesock(sock,buffer); //Processed string sent to client.
     }
    }

    else{
      n = write(sock,"refused",strlen("refused")); if (n < 0) error("
          ERROR writing to socket");
      printf("%s failed to authenticate!\n",username);
      goto auth;

    }
}
```

3.2 Code for ps1_c

```
/* To run the client program
* first run
* 1. make -B
* 2. ./server.out 12345
* 3. Example:  ./client.out anshuman 12345
* Port no provided to client should be same
* as given to server.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h> //for read and write function
void error(char *msg);
int readsock(int sock,char buffer[]);
int writesock(int sock, char buffer[]);
void readstdin(char buffer[]);

void error(char *msg){
    perror(msg);
    exit(0);
}
// function to read from socket
int readsock(int sock,char buffer[]){
```

```
  int n;
  bzero(buffer,256);
  n = read(sock,buffer,255);
  if (n < 0) error("ERROR reading from socket");
  return n;
}
// function to read from standard Input
void readstdin(char buffer[]){

  bzero(buffer,256);
  fgets(buffer,255,stdin);

}
// function to write into socket
int writesock(int sock, char buffer[]){
  int n;
  n = write(sock,buffer,strlen(buffer));if (n < 0) error("ERROR writing
     to socket");
  return n;
}

int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    // Defining structs to store addresss
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) error("ERROR opening socket");
    server = gethostbyname(argv[1]);

    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }

    bzero((char *) &serv_addr, sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;

    bcopy((char *)server->h_addr,(char *)&serv_addr.sin_addr.s_addr,
       server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr))
       < 0)
        error("ERROR connecting");
```

```
    //Username and Password Section
    auth:
    printf("Username : ");
    readstdin(buffer);
    writesock(sockfd,buffer); //Username sent
    printf("Password : ");
    readstdin(buffer);
    writesock(sockfd,buffer); //Password sent

    readsock(sockfd,buffer); printf("%s\n",buffer);//Check login status.

    if (strcmp(buffer,"granted") == 0){

        while (1) {
          printf("Input Query : ");
          readstdin(buffer);
          buffer[strlen(buffer)-1]  = '\0';
          writesock(sockfd,buffer); //Input Query sent
          if (strcmp(buffer,"q") == 0){
            break;
          }
          else {
            readsock(sockfd,buffer);
            printf("Processed Query : %s\n",buffer);
          }

        }

    }

    else if(strcmp(buffer,"refused") == 0){
        printf("Invalid Credentials!\n");
        goto auth;

    }
    else{
      printf("Something went terribly wrong. Try restarting Client.\n" )
          ;
    }

    return 0;
}
```

3.3 Code for ps2_s

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
```

```c
#include <netdb.h>

#define PORT 4950 // define port number.
#define BUFSIZE 1024 // define buffer size.
void error(char *msg)
{
    perror(msg);
    exit(1);
}
// read from sock
int readsock(int sock,char buffer[]){
  int n;
  bzero(buffer,256);
  n = read(sock,buffer,256);
  if (n < 0) error("ERROR reading from socket");
  return n;
}
// write into sock
int writesock(int sock, char buffer[]){
  int n;
  n = write(sock,buffer,strlen(buffer));
  if (n < 0) error("ERROR writing to socket");
  return n;
}
//store chat_history in file.
void store_history(char chat[])
{
        time_t mytime;
        mytime = time(NULL);
        // printf("%s",ctime(&mytime));
        FILE *fs;
        fs = fopen("chat_history.txt","a");
        fprintf(fs, "%s %s", chat,ctime(&mytime));
        // fprintf(fs,"%s",ctime(&mytime));
        // fprintf(fs,"%s\n",chat);
        fclose(fs);
}
//function to braodcast to every user.
void send_to_all(int j, int i, int sockfd, int nbytes_recvd, char *
    recv_buf, fd_set *master)
{

        if (FD_ISSET(j, master)){
                if (j != sockfd && j != i) {
                        if (send(j, recv_buf, nbytes_recvd, 0) == -1) {
                                perror("send");
                        }
                }
        }
}
//receive a new connection or data from clients.
void send_recv(int i, fd_set *master, int sockfd, int fdmax,char *argv
    [])
{
```

```
        int nbytes_recvd, j;
        char recv_buf[BUFSIZE],chat[BUFSIZE];

        if ((nbytes_recvd = recv(i, recv_buf, BUFSIZE, 0)) <= 0) {
                if (nbytes_recvd == 0) {
                        printf("socket %d hung up\n", i);
                }else {
                        perror("recv");
                }
                close(i);
                FD_CLR(i, master);
        }else {
                store_history(recv_buf);
                //
                // char *c = malloc(strlen(recv_buf)+strlen(argv[1])+1);
                // strcpy(c,recv_buf);
          // strcat(c,argv[1]);
                // strcpy(recv_buf,c);
                for(j = 0; j <= fdmax; j++){
                        send_to_all(j, i, sockfd, nbytes_recvd, recv_buf
                            , master );
                }
        }
}
//accept connection from clients.
void connection_accept(fd_set *master, int *fdmax, int sockfd, struct
    sockaddr_in *client_addr)
{
        char username[256],password[256];
        char name[10],pass[10];

        socklen_t addrlen;
        int newsockfd,n;

        addrlen = sizeof(struct sockaddr_in);
        if((newsockfd = accept(sockfd, (struct sockaddr *)client_addr, &
            addrlen)) == -1) {
                perror("accept");
                exit(1);
        }else {
                auth:
    //Read username from socket
    n = readsock(newsockfd,username); username[n-1] = '\0';
    printf("Username provided : %s\n",username);
                //Read password from socket
    n = readsock(newsockfd,password); password[n-1] = '\0';
    printf("Password provided : %s\n",password);
                //Reading username and password from file user_pass
    FILE * fp = fopen("user_pass","r"); fscanf(fp, "%s %s",name,pass);
    // Compare username and password with Credentials from file.
    if (strcmp(username,name) == 0 && strcmp(password,pass) == 0 )
                {

            n = write(newsockfd,"granted",strlen("granted")); if (n < 0)
```

```
                error("ERROR writing to socket");
            printf("%s granted access!\n",username);
                        FD_SET(newsockfd, master);
                        if(newsockfd > *fdmax)
                        {
                                *fdmax = newsockfd;
                        }
                        printf("new connection from %s on port %d \n",
                            inet_ntoa(client_addr->sin_addr), ntohs(
                            client_addr->sin_port));
                }
                else
                {
                        n = write(newsockfd,"refused",strlen("refused"))
                            ; if (n < 0) error("ERROR writing to socket"
                            );
                        printf("%s failed to authenticate!\n",username);
                        goto auth;

                }

                FD_SET(newsockfd, master);
                if(newsockfd > *fdmax){
                        *fdmax = newsockfd;
                }
                printf("new connection from %s on port %d \n",inet_ntoa(
                    client_addr->sin_addr), ntohs(client_addr->sin_port)
                    );
        }
}
//creating socket and binding.
void connect_request(int *sockfd, struct sockaddr_in *my_addr)
{
        int yes = 1;

        if ((*sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                perror("Socket");
                exit(1);
        }

        my_addr->sin_family = AF_INET;
        my_addr->sin_port = htons(4950);
        my_addr->sin_addr.s_addr = INADDR_ANY;
        memset(my_addr->sin_zero, '\0', sizeof my_addr->sin_zero);

        if (setsockopt(*sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(
            int)) == -1) {
                perror("setsockopt");
                exit(1);
        }

        if (bind(*sockfd, (struct sockaddr *)my_addr, sizeof(struct
            sockaddr)) == -1) {
                perror("Unable to bind");
```

```
                exit(1);
        }
        if (listen(*sockfd, 2) == -1) {
                perror("listen");
                exit(1);
        }
        printf("\nTCPServer Waiting for client on port 4950\n");
        fflush(stdout);
}
int main(int argc, char *argv[])
{
        fd_set master;
        fd_set read_fds;
        int fdmax, i;
        int sockfd= 0;
        struct sockaddr_in my_addr, client_addr;

        FD_ZERO(&master);
        FD_ZERO(&read_fds);
        connect_request(&sockfd, &my_addr);
        FD_SET(sockfd, &master);

        fdmax = sockfd;
        while(1){
                read_fds = master;
                if(select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1){
                        perror("select");
                        exit(4);
                }

                for (i = 0; i <= fdmax; i++){
                        if (FD_ISSET(i, &read_fds)){
                                if (i == sockfd)
                                        connection_accept(&master, &
                                            fdmax, sockfd, &client_addr)
                                            ;
                                else
                                        send_recv(i, &master, sockfd,
                                            fdmax,argv);
                        }
                }
        }
        return 0;
}
```

3.4 Code for ps2_c

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```c
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define PORT 4950 // define port number.
#define BUFSIZE 1024 // define buffer size.
void error(char *msg)
{
    perror(msg);
    exit(1);
}
// read from sock
int readsock(int sock,char buffer[]){
  int n;
  bzero(buffer,256);
  n = read(sock,buffer,256);
  if (n < 0) error("ERROR reading from socket");
  return n;
}
// write into sock
int writesock(int sock, char buffer[]){
  int n;
  n = write(sock,buffer,strlen(buffer));
  if (n < 0) error("ERROR writing to socket");
  return n;
}
//store chat_history in file.
void store_history(char chat[])
{
        time_t mytime;
        mytime = time(NULL);
        // printf("%s",ctime(&mytime));
        FILE *fs;
        fs = fopen("chat_history.txt","a");
        fprintf(fs, "%s %s", chat,ctime(&mytime));
        // fprintf(fs,"%s",ctime(&mytime));
        // fprintf(fs,"%s\n",chat);
        fclose(fs);
}
//function to braodcast to every user.
void send_to_all(int j, int i, int sockfd, int nbytes_recvd, char *
   recv_buf, fd_set *master)
{

        if (FD_ISSET(j, master)){
                if (j != sockfd && j != i) {
                        if (send(j, recv_buf, nbytes_recvd, 0) == -1) {
                                perror("send");
                        }
                }
        }
}
//receive a new connection or data from clients.
void send_recv(int i, fd_set *master, int sockfd, int fdmax,char *argv
```

```
    [])
{
        int nbytes_recvd, j;
        char recv_buf[BUFSIZE],chat[BUFSIZE];

        if ((nbytes_recvd = recv(i, recv_buf, BUFSIZE, 0)) <= 0) {
                if (nbytes_recvd == 0) {
                        printf("socket %d hung up\n", i);
                }else {
                        perror("recv");
                }
                close(i);
                FD_CLR(i, master);
        }else {
                store_history(recv_buf);
                //
                // char *c = malloc(strlen(recv_buf)+strlen(argv[1])+1);
                // strcpy(c,recv_buf);
          // strcat(c,argv[1]);
                // strcpy(recv_buf,c);
                for(j = 0; j <= fdmax; j++){
                        send_to_all(j, i, sockfd, nbytes_recvd, recv_buf
                            , master );
                }
        }
}
//accept connection from clients.
void connection_accept(fd_set *master, int *fdmax, int sockfd, struct
    sockaddr_in *client_addr)
{
        char username[256],password[256];
        char name[10],pass[10];

        socklen_t addrlen;
        int newsockfd,n;

        addrlen = sizeof(struct sockaddr_in);
        if((newsockfd = accept(sockfd, (struct sockaddr *)client_addr, &
            addrlen)) == -1) {
                perror("accept");
                exit(1);
        }else {
                auth:
    //Read username from socket
    n = readsock(newsockfd,username); username[n-1] = '\0';
    printf("Username provided : %s\n",username);
                //Read password from socket
    n = readsock(newsockfd,password); password[n-1] = '\0';
    printf("Password provided : %s\n",password);
                //Reading username and password from file user_pass
    FILE * fp = fopen("user_pass","r"); fscanf(fp, "%s %s",name,pass);
    // Compare username and password with Credentials from file.
    if (strcmp(username,name) == 0 && strcmp(password,pass) == 0 )
                {
```

```
            n = write(newsockfd,"granted",strlen("granted")); if (n < 0)
                 error("ERROR writing to socket");
            printf("%s granted access!\n",username);
                        FD_SET(newsockfd, master);
                        if(newsockfd > *fdmax)
                        {
                                *fdmax = newsockfd;
                        }
                        printf("new connection from %s on port %d \n",
                            inet_ntoa(client_addr->sin_addr), ntohs(
                            client_addr->sin_port));
                }
                else
                {
                        n = write(newsockfd,"refused",strlen("refused"))
                            ; if (n < 0) error("ERROR writing to socket"
                            );
                        printf("%s failed to authenticate!\n",username);
                        goto auth;

                }

                FD_SET(newsockfd, master);
                if(newsockfd > *fdmax){
                        *fdmax = newsockfd;
                }
                printf("new connection from %s on port %d \n",inet_ntoa(
                    client_addr->sin_addr), ntohs(client_addr->sin_port)
                    );
        }
}
//creating socket and binding.
void connect_request(int *sockfd, struct sockaddr_in *my_addr)
{
        int yes = 1;

        if ((*sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
                perror("Socket");
                exit(1);
        }

        my_addr->sin_family = AF_INET;
        my_addr->sin_port = htons(4950);
        my_addr->sin_addr.s_addr = INADDR_ANY;
        memset(my_addr->sin_zero, '\0', sizeof my_addr->sin_zero);

        if (setsockopt(*sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(
            int)) == -1) {
                perror("setsockopt");
                exit(1);
        }

        if (bind(*sockfd, (struct sockaddr *)my_addr, sizeof(struct
```

```
            sockaddr)) == -1) {
                perror("Unable to bind");
                exit(1);
        }
        if (listen(*sockfd, 2) == -1) {
                perror("listen");
                exit(1);
        }
        printf("\nTCPServer Waiting for client on port 4950\n");
        fflush(stdout);
}
int main(int argc, char *argv[])
{
        fd_set master;
        fd_set read_fds;
        int fdmax, i;
        int sockfd= 0;
        struct sockaddr_in my_addr, client_addr;

        FD_ZERO(&master);
        FD_ZERO(&read_fds);
        connect_request(&sockfd, &my_addr);
        FD_SET(sockfd, &master);

        fdmax = sockfd;
        while(1){
                read_fds = master;
                if(select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1){
                        perror("select");
                        exit(4);
                }

                for (i = 0; i <= fdmax; i++){
                        if (FD_ISSET(i, &read_fds)){
                                if (i == sockfd)
                                        connection_accept(&master, &
                                            fdmax, sockfd, &client_addr)
                                            ;
                                else
                                        send_recv(i, &master, sockfd,
                                            fdmax,argv);
                        }
                }
        }
        return 0;
}
```

References

[1] Beef.us. *Beej's Guide to Network Programming*. `http://beej.us/guide/bgnet/`.

[2] GeeksforGeeks. *fork() in C*. `https://www.geeksforgeeks.org/fork-system-call/`.

[3] wikipedia.org. *Network socket*. https://en.wikipedia.org/wiki/Network_socket.