NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

Cachar, Assam

B.Tech. Vth Sem

Subject Code: CS-311

Subject Name: Computer Network Laboratory

Submitted By:

Name : Subhojit Ghimire

Sch. Id. : 1912160

Branch : CSE – B

Q.7. Write a program for "Remote Command Execution" using sockets. (Description: The client sends command(s) and data parameters (if required) to the server. The server will execute the command and send back the result to the client. Execute 3 commands.)

AIM: TO IMPLEMENT "REMOTE COMMAND EXECUTION" USING UDP IN CPP.

THEORY: 1. REMOTE COMMAND EXECUTION: Remote Command or Code Execution (RCE) is when external code is able to execute internal, operating-system level commands on a server from a distance. Also known as Arbitrary Code Execution, RCE is a concept that describes a form of cyber-attack in which the attacker can solely command the operation of another person's computing device or computer.

2. UDP CLIENT SERVER: In UDP, the client does not form a connection with the server like in TCP. Instead, the client just sends a datagram. Similarly, the server does not need to accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.

CODE:

```
// RCE SERVER

#include <iostream>
#include <cstdlib>
#include <cstring>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <fstream>

#define PORT 1194
#define MAXLINE 1024

using namespace std;

int main() {
    int sockfd;
    struct sockaddr_in servaddr, cliaddr;
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
        perror("FAIL: SOCKET CREATION\n");
        exit(EXIT_FAILURE);
    }
    cout << "SUCCESS: SOCKET CREATED\n";
    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
```

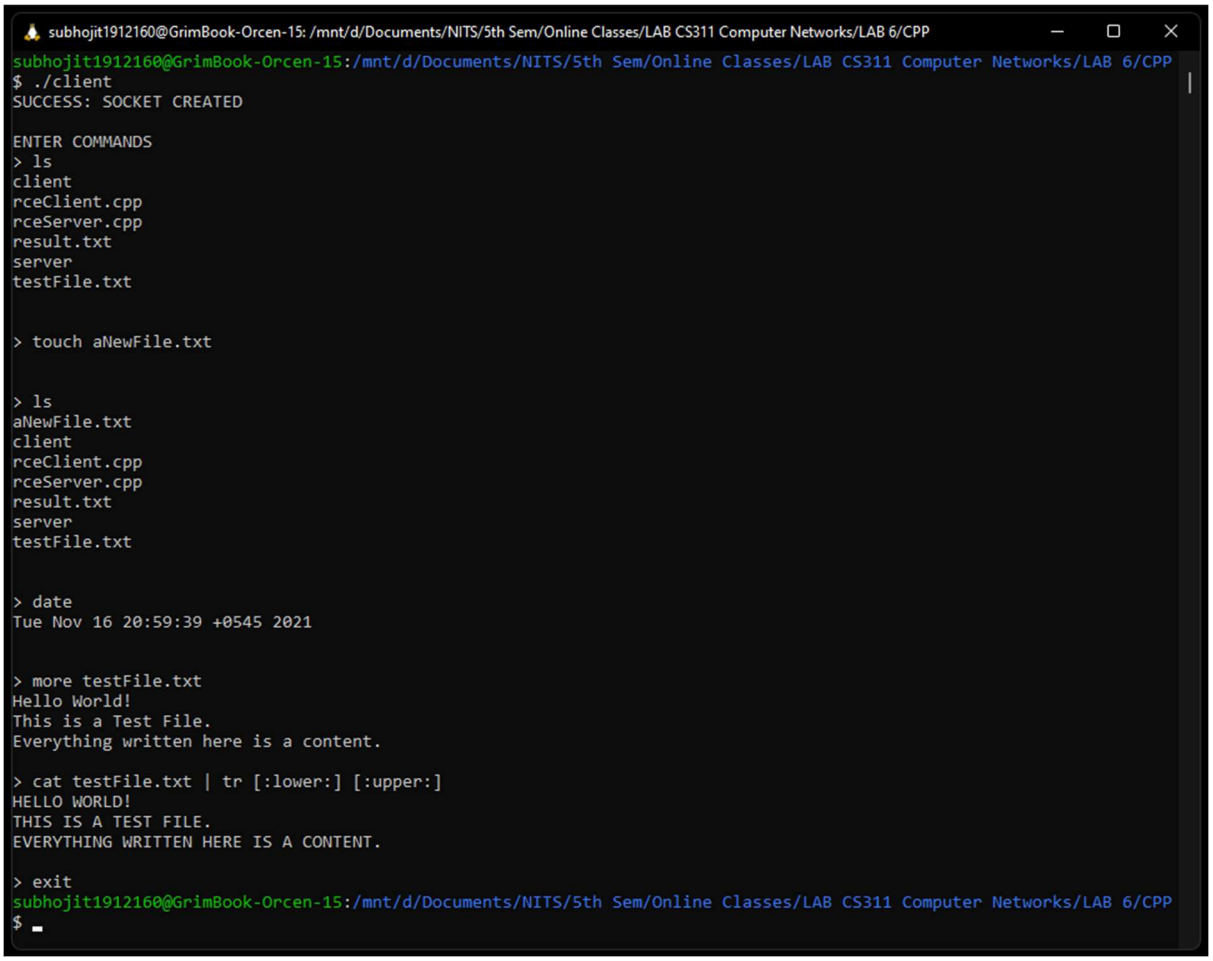
```
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons (PORT);

if (bind (sockfd, (const struct sockaddr *) &servaddr,
        sizeof (servaddr)) < 0 ) {
    perror ("FAIL : SERVER BIND \n");
    exit ( EXIT_FAILURE );
}
cout << "SUCCESS : SERVER BOUND \n";

cout << "SERVER OPEN FOR INCOMING MESSAGES \n\n";
char buffer [MAXLINE];
char command [MAXLINE+15];
unsigned int len, nn;
len = sizeof (cliaddr);
while (1) {
    memset (buffer, 0, MAXLINE);
    bzero (command, MAXLINE);
    nn = recvfrom (sockfd, (char *) buffer,
        MAXLINE, MSG_WAITALL, (struct
        sockaddr *) &cliaddr, &len );
    buffer [nn] = '\0';
    cout << "REQUEST RECEIVED. EXECUTING
        COMMAND: " << buffer;
    strcpy (command, buffer);
    nn = strlen (command);
    command [nn-1] = '\0';
    strcat (command, " > result.txt ");
```

```
        system (command);
        FILE *ff = fopen ("result.txt", "r");
        bzero (buffer, MAXLINE);
        for (nn=0 ; !feof (ff) ; ++nn)
                buffer [nn] = getc (ff);
        buffer [nn-1] = '\0';
        cout << "SUCCESS: RESULT SENT BACK TO CLIENT";
        cout << endl << endl;
        sendto (sockfd, (const char *) buffer, strlen(buffer),
                MSG_CONFIRM, (const struct sockaddr *)
                &cliaddr, len);
    }
    close (sockfd);
    return 0;
}

// RCE CLIENT

# include <iostream>
# include <cstdlib>
# include <cstring>
# include <unistd.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <arpa/inet.h>
# include <netinet/in.h>

# define PORT 1194
# define MAXLINE 1024
```

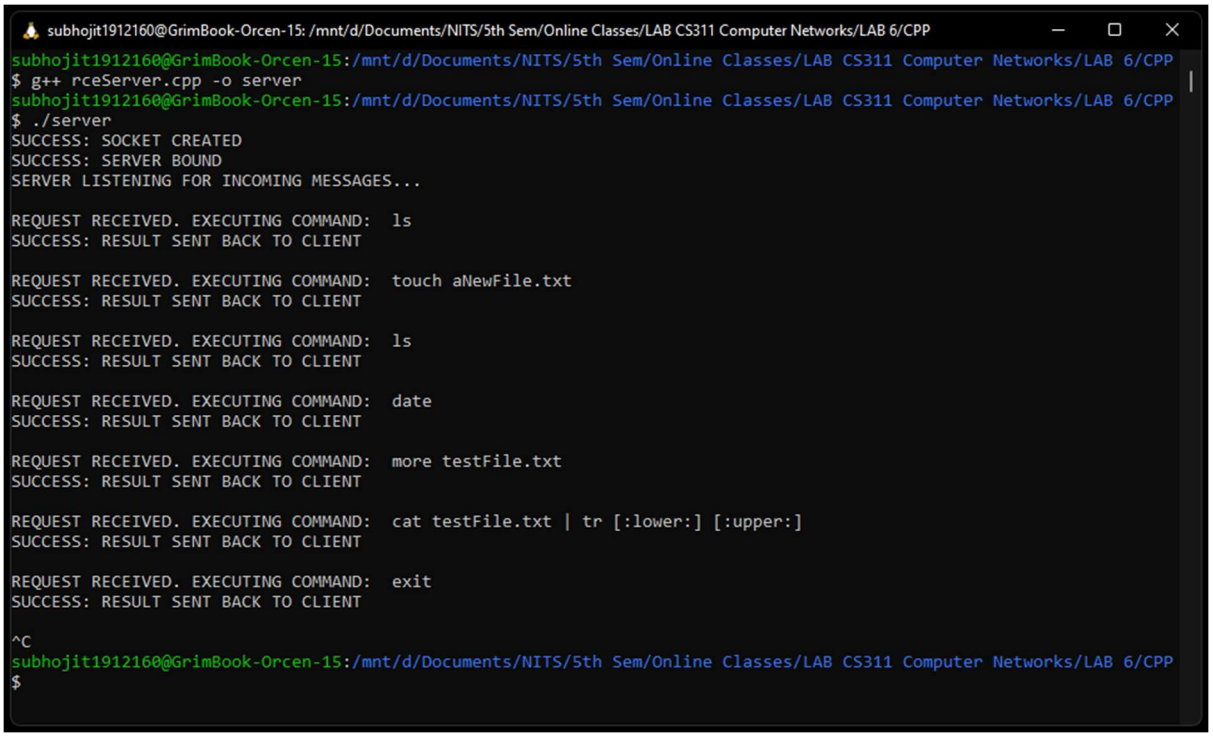
```
using namespace std;
int main () {
    int sockfd;
    struct sockaddr_in servaddr;
    if ((sockfd = socket (AF_INET, SOCK_DGRAM, 0)) <0))
            exit (EXIT_FAILURE);
    cout << "SUCCESS : SOCKET CREATED \n\n";
    memset (&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons (PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;
    char buffer [MAXLINE], command [MAXLINE];
     cout << "ENTER COMMANDS";
    unsigned int nn, len;
    while (1) {
        memset (command, 0, MAXLINE);
        memset (buffer, 0, MAXLINE);
        cout << "\n> " ;
        fgets (command, MAXLINE, stdin);
        sendto (sockfd, (const char *) command, strlen
            (command), MSG_CONFIRM, (const struct
             sockaddr *) &servaddr, sizeof (servaddr));
       nn = recvfrom (sockfd, (char *) buffer, MAXLINE,
            MSG_WAITALL, (struct sockaddr *)&servaddr,
             &len);
       buffer [nn] = '\0';
       if (strncmp (command, "exit", 4) == 0)
              break;
       cout << buffer  <<endl;
}
close (sockfd); return 0;
}
```

OUTPUT:

// RCE CLIENT



// RCE SERVER



Output Explanation:

Firstly, the Remote Command Execution Server and Client are compiled and run. After the connection is established, in the client terminal, the command to be executed is entered. This command is sent to the server as a message request. The server then runs the requested command, saves the result in buffer and sends the buffer back to client. The client then displays the buffer sent by the server, which is actually the result of the command sent by the client machine. For example, as seen in the output images above, the client sends “ls” as the command request. The server then executes “ls” command on its machine and sends back the result to client. The client then displays the result, which, in this case, is listing of all files present in the currently open folder location of the server. Similarly, client send the command “touch aNewFile.txt”. The server receives the request and executes the command. This “touch” command line utility is used to create a new file. So, in the third time, when the client requests to see all the listed files, the newly created file is also displayed, as it gets created in the second command. Similarly, the other commands are requested by the client, the server executed the command on its machine and sends the result back to the client, which is then displayed on the client terminal.