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.4. Write an "Echo Client" and "Echo Server" using UDP to estimate the round trip time from client to server. The server should be such that it can accept multiple connections at any given time. (Description: Multiple clients connected at the same time to one server for connection establishment, server has to listen to all the clients, use I/O monitoring calls if required.)

AIM: To Implement "Echo Client Server" To Estimate The Round Trip Time From Client To Server Using UDP In CPP.

THEORY: 1. Echo Client Server: It is an application that allows a client and a server to connect so a client can send a message to the server and the server can receive the message and send, or echo, it back to the client.

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.

3. Round Trip Time: The RTT is the duration in milliseconds (ms) taken by a network request to go from a starting point to a destination and back again to the starting point.

4. I/O MULTIPLEXING : It essentially means reading from or writing to multiple file descriptors simultaneously. I/O multiplexing is used when a client is handling multiple descriptors or multiple sockets, as well as when a server handles both TCP and UDP or handles multiple services and protocols.

The entire operation can be broken down as follows:

UDP ECHO CLIENT:

(i) A socket is created and binded.

(ii) The messages are sent from the user using sendto() function and the RTT start time is recorded.

(iii) The data from the server is received using recvfrom() and the RTT end time is recorded.

(iv) The final RTT time is displayed along with the echo message.

UDP ECHO SERVER:

(i) A socket is created and binded to an advertised port number.

(ii) An infinite loop is started to process the client requests for connections.

(iii) The process receives data from the client using recvfrom() function and echoes the same data using the sendto() function.

(iv) This server is capable of handling multiple clients automatically as UDP is a datagram based protocol and hence, no exclusion connection is required to a client.

CODE :

// ECHO 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>

#define PORT 8080
#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 BINDING \n");
        exit (EXIT_FAILURE);
}
cout << " SUCCESS: SERVER BOUND \n";

cout << " SERVER LISTENING FOR MESSAGES TO ECHO BACK...\n";
char buffer [MAXLINE];
unsigned int len, nn;
len = sizeof (cliaddr);
while (1) {
        memset (buffer, 0, MAXLINE);
        nn = recvfrom (sockfd, (char *) buffer, MAXLINE,
                MSG_WAITALL, (struct sockaddr *)&cliaddr,
                &len );
        buffer[nn] = '\0' ;
        cout << "\n" << buffer << " : " ;
        memset (buffer, 0, MAXLINE);
        nn = recvfrom (sockfd, (char *) buffer, MAXLINE,
                MSG_WAITALL, (struct sockaddr *) &cliaddr,
                &len);
        buffer [nn] = '\0' ;
        cout << buffer;
```

```
        sendto (sockfd, (const char *) buffer, strlen(buffer),
                MSG_CONFIRM, (const struct sockaddr *) &cliaddr,
                len);
        cout << "ECHO SENT.\n";
    }
    close (sockfd);
    return 0;
}
```

// ECHO CLIENT

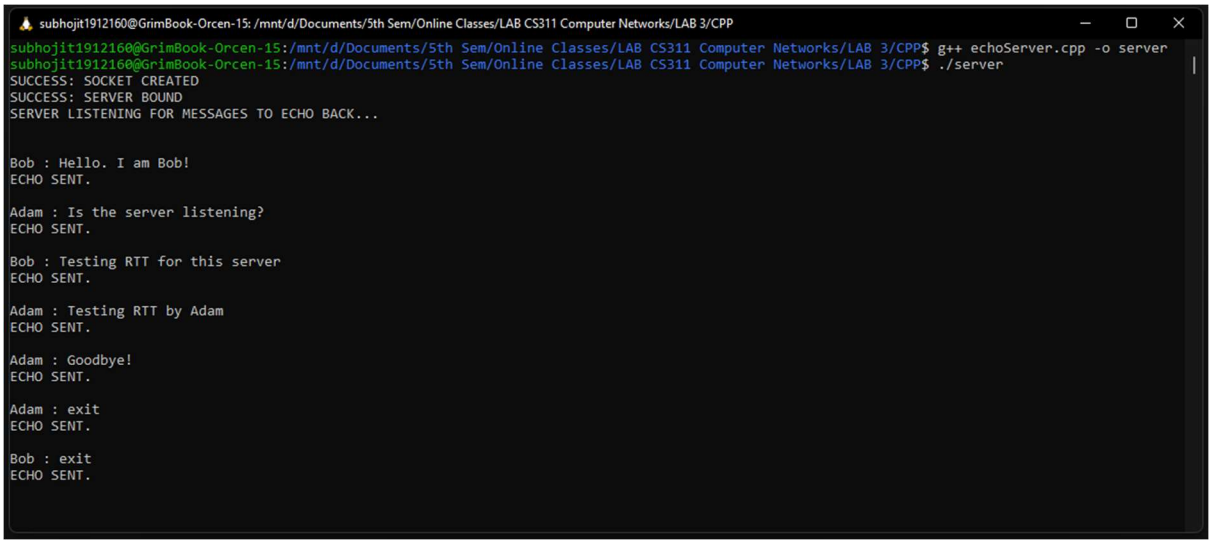
```
# include <iostream>
# include <cstdlib>
# include <unistd.h>
# include <cstring>
# include <sys/types.h>
# include <sys/socket.h>
# include <arpa/inet.h>
# include <netinet/in.h>
# include <ctime>

# define PORT 8080
# define MAXLINE 1024

using namespace std;
```

```
int main (int argc, char *argv[]) {
        clock_t start, end;
        double cpuTimeUsed;
        int sockfd;
        struct sockaddr_in servaddr;
        if ((sockfd = socket (AF_INET, SOCK_DGRAM, 0))
                                                < 0) {
                perror ("FAIL: SOCKET CREATION \n");
                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];
        char msg [MAXLINE];
        char clientName [10];
        strcpy (clientName, argv [1]);
        cout << " SEND MESSAGES \n";

        unsigned int nn, len;
        while (1) {
                memset (msg, 0, MAXLINE);
                memset (buffer, 0, MAXLINE);
                cout << " > ";
                fgets (msg, MAXLINE, stdin);
```

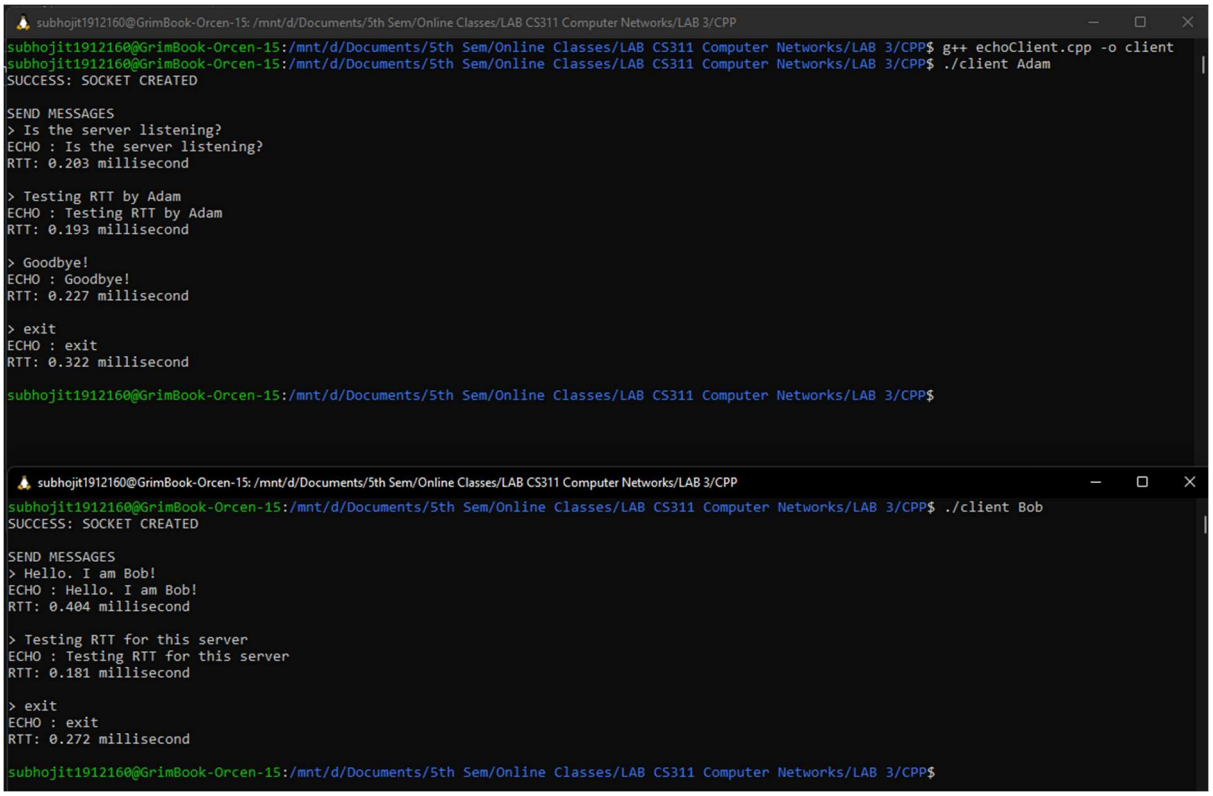
```
start = clock();
sendto (sockfd, (const char *) clientName, strlen(clientName)
        , MSG_CONFIRM, (const struct sockaddr *) &servaddr,
        sizeof(servaddr));
sendto (sockfd, (const char *) msg, strlen(msg),
        MSG_CONFIRM, (const struct sockaddr *) &servaddr,
        sizeof(servaddr));
nn = recvfrom (sockfd, (char *) buffer, MAXLINE,
     MSG_WAITALL, (struct sockaddr *) &servaddr,
     &len);
buffer[nn] = '\0';
sleep(1);
end = clock();

cout << "ECHO : " << buffer;
cpuTimeUsed = ((double)(end-start)) / CLOCKS_PER_SEC;
cout << "RTT : " <<(cpuTimeUsed * 1000) << " milliseconds";
cout << endl << endl;

if (strncmp(msg, "exit", 4) == 0)
        break;
}

close(sockfd);
return 0;
}
```

OUTPUT:

// UDP ECHO SERVER



// UDP ECHO CLIENT



Output Explanation:

Firstly, the UDP Server programme is compiled and run. The server, after socket creation and binding, is open to all the incoming connections. Then, the UDP Client programme is compiled and run in two different terminals, each terminal acting as its own separate client. Either of the client can send message at any time; in the above screenshot, the client named “Bob” sends the first message which is echoed back and the round trip time for the message from client to server and back to client is displayed. Similarly, the client named “Adam” send their own message and the RTT for the message to transmit from the client to server and back to client is displayed. In the server terminal as well, every log of received message from all the client and echoed messages are recorded. In the end, the clients break the connection with the server by sending “exit” message, which terminates the processes.