**Submitted by – Paras Jain (2018KUCP1006)**

# Computer Networks Lab Assignment - 4 Socket Programming: Concurrent Client Server Program

# 1. Use fork () in your program as an additional function.

# 2. Concurrent server will handle multiple clients at the same time unlike iterative client server.

**Server Side Code:**

//Server Side Code

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <sys/types.h>

#include <unistd.h>

#include <time.h>

int main(){

    int mySocket, clintConnt, clintNum = 0;

    char Send\_buffer[1024], Rec\_buffer[1024];

    struct sockaddr\_in ipOfServer;

    //Creating the socket, arguments are: Internet domain, Stream socket, Default protocol (TCP)

    mySocket = socket(AF\_INET, SOCK\_STREAM, 0);

    //Configure settings of the server address struct

    ipOfServer.sin\_family = AF\_INET;

    // Set port number, using htons function to use proper byte order

    ipOfServer.sin\_port = htons(2017);

    ipOfServer.sin\_addr.s\_addr = htonl(INADDR\_ANY); //bind to any local address

    // Set all bits of the padding field to 0

    memset(ipOfServer.sin\_zero, '\0', sizeof ipOfServer.sin\_zero);

    // Bind the address struct to the socket

    bind(mySocket, (struct sockaddr \*)&ipOfServer, sizeof(ipOfServer));

    //Listen on the socket, with 20 max connection requests queued

    listen(mySocket, 20);

    time\_t clock; //for capturing timestamp

    while (1){

        //Accept call creates a new socket for the incoming connection

        clintConnt = accept(mySocket, (struct sockaddr \*)NULL, NULL);

        clock = time(NULL); //Capturing time when a client hits

        clintNum++;         //Incrementing client number

        if (fork() > 0)

        { // for parent process this code will execute to serve the current client

            char timStmp[100];

            snprintf(timStmp, sizeof(timStmp), "%.24s\r", ctime(&clock));

            printf("clint number %d hit at time: %s\n", clintNum, timStmp);

            //Recieve message from the socket of the incoming connection

            recv(clintConnt, Rec\_buffer, 1024, 0);

            clock = time(NULL);

            snprintf(timStmp, sizeof(timStmp), "%.24s\r", ctime(&clock));

            //Printing recieved message

            printf("Msg recieved from client number %d at time: %s\n%s\n", clintNum, timStmp, Rec\_buffer);

            //Generating msg for sending

            sprintf(Send\_buffer, "Client number %d your msg recieved at time %s", clintNum, timStmp);

            //Sending message to the socket of the incoming connection

            send(clintConnt, Send\_buffer, 1024, 0);

        }

        else

        {

            continue; // for child process we want the while loop to run again so that it can wait for a new client.

        }

    }

    return 0;

}

**Client Side Code:**

#include <stdio.h>

#include <string.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <netinet/in.h>

int main(){

    int clientSocket;

    char Send\_buffer[1024], Rec\_buffer[1024];

    struct sockaddr\_in ipOfServer;

    //Creating the socket, arguments are: Internet domain, Stream socket, Default protocol (TCP)

    clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);

    //Configure settings of the server address struct

    ipOfServer.sin\_family = AF\_INET;

    //Set port number, using htons function to use proper byte order

    ipOfServer.sin\_port = htons(2017);

    //Set IP address to localhost

    ipOfServer.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    //Set all bits of the padding field to 0

    memset(ipOfServer.sin\_zero, '\0', sizeof ipOfServer.sin\_zero);

    //Connect the socket to the server using the address struct

    if (connect(clientSocket, (struct sockaddr \*)&ipOfServer, sizeof(ipOfServer)) < 0){

        //connecting to server, -ve value implies unsuccessful

        printf("Connection failed due to port and ip problems\n");

        return 1;

    }

    printf("Enter msg\n");

    gets(Send\_buffer);

    // scanf("%s",Send\_buffer);

    send(clientSocket, Send\_buffer, 1024, 0);

    //Read the message from the server into the buffer

    recv(clientSocket, Rec\_buffer, 1024, 0);

    //Print the received message

    printf("Msg from server:\n");

    puts(Rec\_buffer);

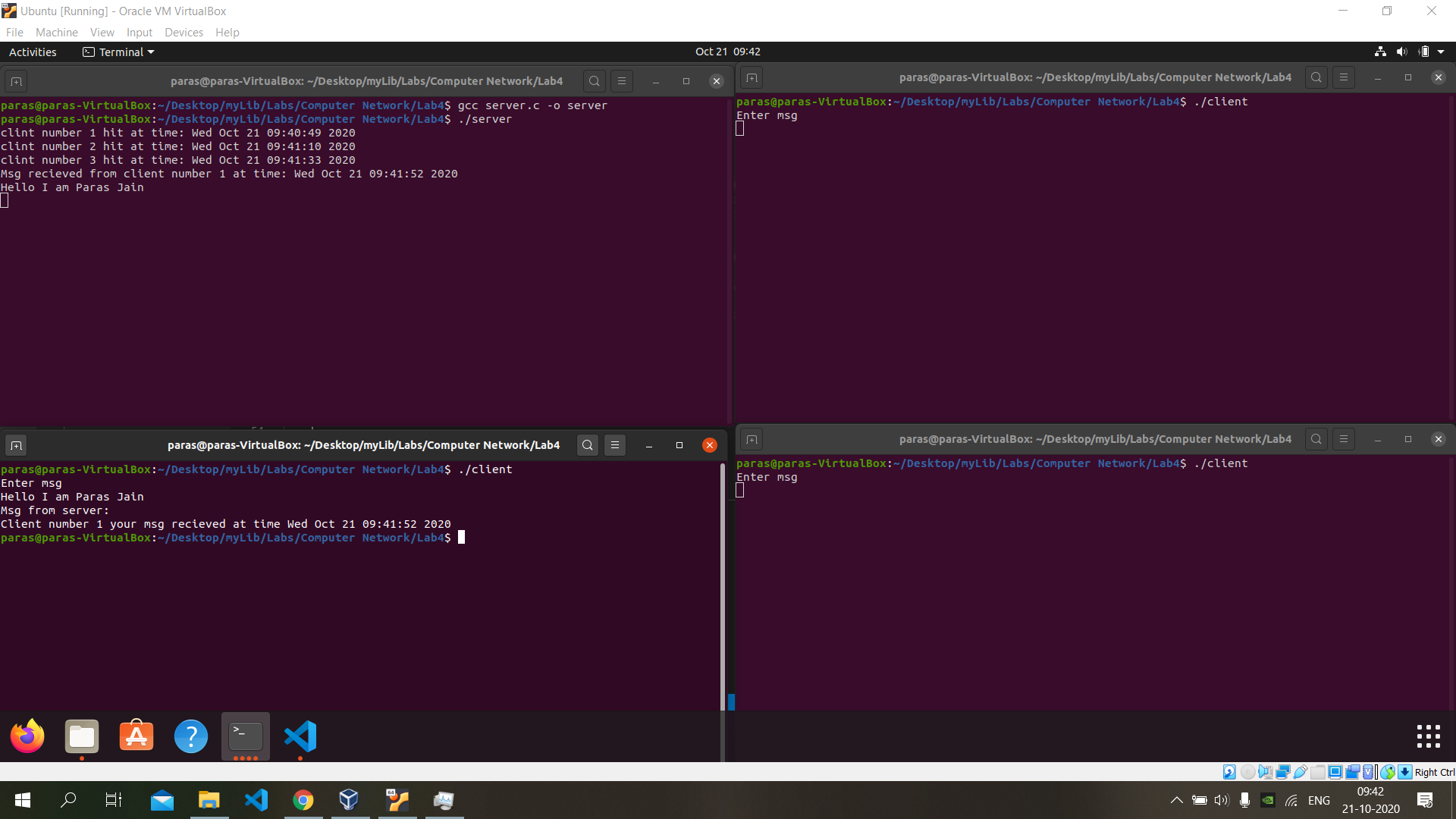
    return 0;

}

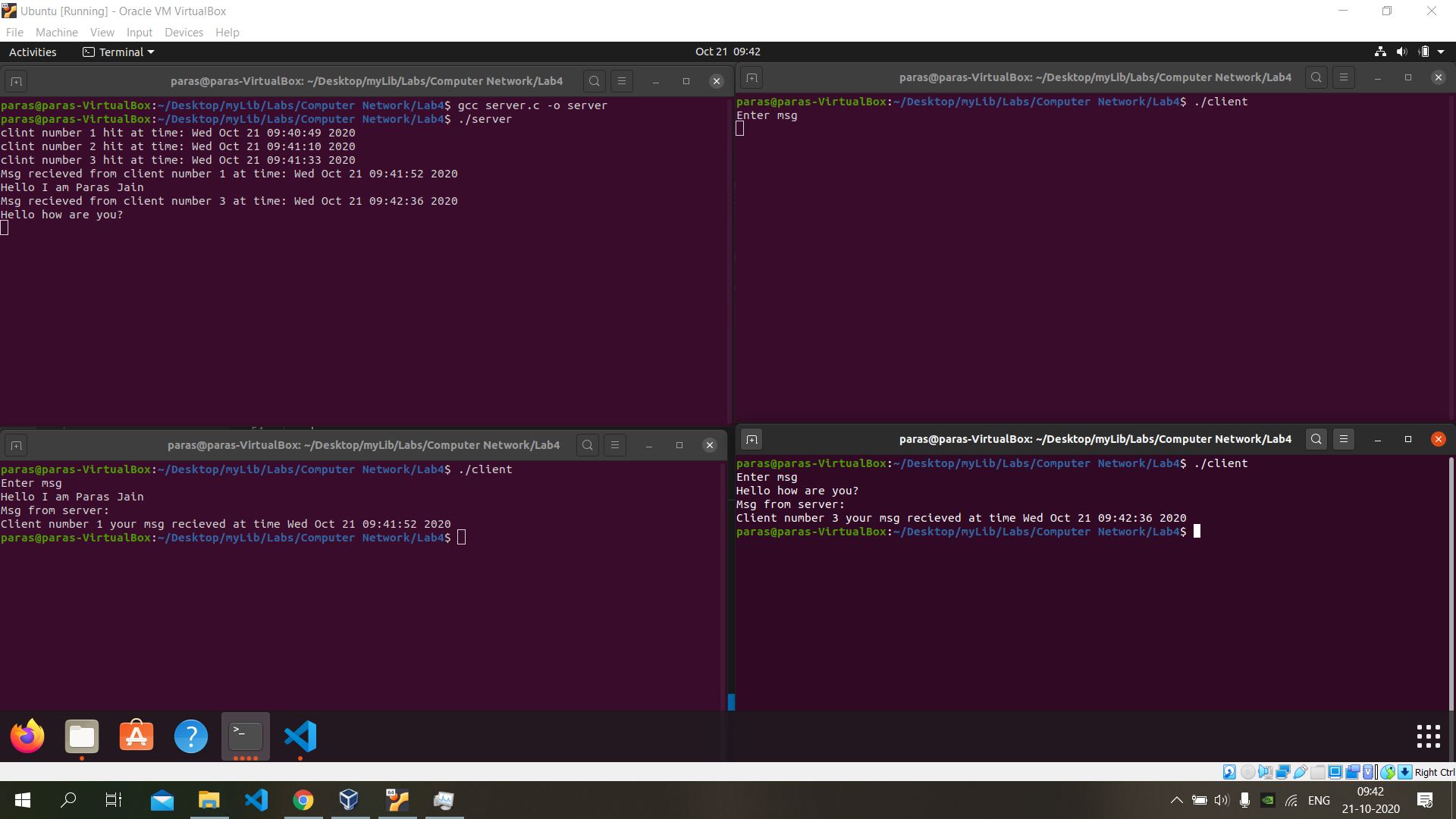
In this I executed 3 clients. If it were iterative then once a client is being served at that time none other client can hit, but here 3 clients have hit even before client 1 completed its request.

Output:

3 clients are running simultaneously and server received their request. Client 1 sent message. It is also showing time at which each client started.



Now client 3 sent message



Now client 2 sent message

