ONLINE QUIZ SYSTEM

A COURSE PROJECT REPORT

By

Nitin Desai (RA2011030010125)
Utkarsh Ajmani (RA2011030010130)
Manubhav Sharma (RA2011030010159)

Under the guidance of

Dr. Prasath N

Associate Professor, Networks and Communication

In partial fulfilment for the Course

of

18CSC302J - COMPUTER NETWORKS

in Networking And Communications

FACULTY OF ENGINEERING AND TECHNOLOGY SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chengalpattu District

NOVEMBER 2022

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report "**Online Quiz System**" is the bonafide work of **Nitin Desai(RA2011030010125)**, **Utkarsh Ajmani(RA2011030010130)** and **Manubhav Sharma(RA2011030010159)** who carried out the project work under my supervision.

SIGNATURE

Dr. Prasath N
Associate
Professor
Department of Networking And
CommunicationsSRM Institute of Science
and Technology

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V. Gopal,** for bringing out novelty in all executions.

We would like to express our heartfelt thanks to the Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express our sincere thanks to **Course Audit Professor Dr. Annapurani Panaiyappan, Professor and Head, Department of Networking and Communications,** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our Course project Faculty **Dr. Prasath N, Associate Professor, Networking And Communications,** for his/her assistance, timely suggestions, and guidance throughout the duration of this course project.

We extend my gratitude to our **Dr.Annapurani Panaiyappan, Professor and Head, Networking And Communications,** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

ABSTRACT

A network has to be designed for an organization in such a way that a web browser and local PCs are made An application for web browsers is hosted by the company on a server that is available to other network hosts(engineers) as well as the permitted (Sales Manager) and not other hosts in the network (Salesman).

On the company's network, the department routers are interconnected so that users may access the server without being blocked. A network for the same was designed using Cisco Packet Tracer version 8.0.0. The requirements were emulated and tested for connectivity.

Table of Contents

1. ABSTRACT

Online Quiz System is a web-based examination system where the quiz is taken online, i.e., through the internet or intranet using a computer system. The Online Quiz System aims to conduct Quizzes efficiently and conserve efforts put in for assessment. The main objective of the Online Quiz System is to efficiently evaluate the candidate through a fully automated system that saves much time and gives fast results. Quizzes can be administered using the Online Quiz System. A teacher has control of the question bank and is supposed to configure the question bank for the quiz. The system carries out the test and auto-grading for questions which is fed into the system. Administrative control of the whole system is provided. Online Quiz System aims to focus on creating a practical quiz experience. We employ socket programming and multithreading to generate a quiz environment that can easily host multiple clients with this project. The administrators and participants attempting the online quiz can communicate with the system through this project, thus facilitating effective implementation and monitoring of various activities of Online Quiz like conducting the quiz, generating questions and accepting the responses. The introduction of online quiz software can replace the conventional system of assessment. Various quiz running agencies can now perform and host many participants freely and cost-effectively through computer-based tests.

2. INTRODUCTION

As modern organizations are automated, and computers are working as per the instructions, it becomes essential to coordinate human beings, commodities and computers in a current organization. The administrators and participants attempting the online quiz can communicate with the system through this project, thus facilitating effective implementation and monitoring of various activities of Online Quiz like conducting the quiz, generating questions and accepting the responses. Technological advancements in this era of digitization and is a boon to the world have also been advantageous to the educational sector.

The introduction of online quiz software can replace the conventional system of assessment. Various quiz conducting agencies can now perform and host many participants freely and cost-effectively through computer-based tests. Quizzes can be administered using the Online Quiz System. A teacher has control of the question bank and is supposed to configure the question bank for the quiz. The system carries out the test and auto-grading for questions which is fed into the system. Administrative control of the whole system is provided. Online Quiz System aims to focus on creating a practical quiz experience. We employ socket programming and multithreading to generate a quiz environment that can easily host multiple clients with this project.

a. TCP/IP - TCP/IP stands for Transmission Control Protocol/Internet Protocol and is a suite of communication protocols used to interconnect network devices on the internet. TCP/IP is also used as a communications protocol in a private computer network (an intranet or extranet). TCP/IP specifies how data is exchanged over the internet by providing end-to-end communications that identify how it should be broken into packets, addressed, transmitted, routed and received at the destination. TCP/IP requires little central management and is designed to make networks reliable to recover automatically from the failure of any device on the web. TCP/IP uses the client-server model of communication. A user or machine (a client) is provided with a service (like sending a webpage) by another computer (a server) in the network. The advantages of using the TCP/IP model include the following:

   i. helps establish a connection between different types of computers;
   ii. works independently of the OS;
   iii. supports many routing protocols;
   iv. uses a client-server architecture that is highly scalable;
   v. can be operated separately;
   vi. supports several routing protocols; and
   vii. is lightweight and doesn't place unnecessary strain on a network or computer.

b. Multithreading - Multithreading is a specialized form of multitasking. Multitasking is the feature that allows your computer to run two or more programs concurrently. In general, there are two types of multitasking: process-based and thread-based. Process-based multitasking handles the concurrent execution of programs. Thread-based multitasking deals with the simultaneous performance of pieces of the same program. A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. C does not contain any built-in support for multithreaded applications. Instead, it relies entirely upon the operating system to provide this feature. POSIX Threads or Pthreads provides API available on many Unix-like POSIX systems such as FreeBSD, NetBSD, GNU/Linux, Mac OS X and Solaris. The maximum number of threads that a process may create is implementation-dependent. Once created, threads are peers and may make other threads. There is no implied hierarchy or dependency between threads.

3. REQUIREMENT ANALYSIS

a. Hardware Requirements -

Processor: Intel Core Duo or Higher

RAM: 1 GB

Hard Disk: 500 MB (Minimum free space)

b. Software Requirements -

i. Client Side

Operating System: Windows 7 or higher

Web Browser: Google Chrome

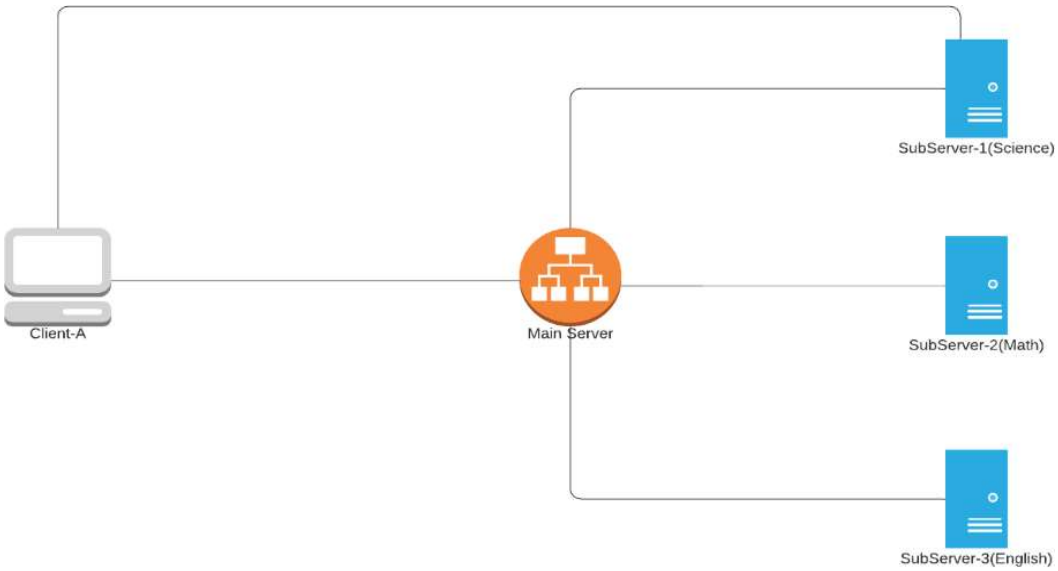
ii. Server Side

Operating Server: Windows 7 or higher

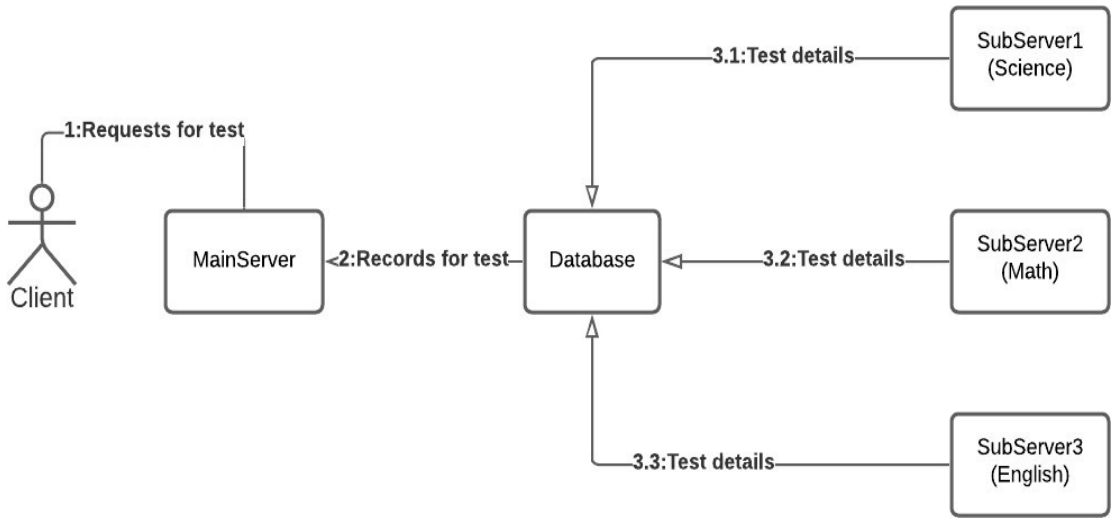
Database Server: SQL Server '08

4. ARCHITECTURE AND DESIGN

I. Architecture Diagram



II. Communication Diagram



a. Design

There will be a main server and three sub-servers (for three different tests). The main server does not want to overburden itself by communicating with all the incoming clients and handle the tests, so it only stores the info about which sub-server contains what sort of test (Three sub-servers each of which contains one type of test i.e., Science, Math or English). When a sub-server connects to the main server it sends the main server a string containing the Test Name which it can carry out along with its IP and Port number (e.g., Mathematics, Sub-server IP, Sub-server Port). Whenever a client connects to the main server, it will display three kinds of tests (Science, Mathematics and English) to the client. The client will then choose the type of test (e.g., Math). The main server will then send the information of corresponding sub-server to the client. The client will then connect to the corresponding sub-server. The sub-server will then show different types of the corresponding test (e.g., Geometry, Algebra and IQ for Math Sub-server) to the client. The client will select the test type and complete the test. Then sub-server will send the information related to client (Client IP, Client Port, Test Name, Test Type, Marks) back to main server and also to the client. The client will then terminate. The main server will store the information within a file for all the clients.

b. Procedure

i. Design a Main Server.

ii. Design a Client Server.

iii. Design 3 TCP Sub-servers - The string a sub-server send to the main server will have the following format: **Test name, Sub-server IP, Sub-server Port**

iv. Each sub-server will connect to main server. Main server will make a database about the sub-server information it gets from the sub-servers.

v. When client connects with main server and makes a request to give a specific test (e.g., Math), main server will search the record in the database it has for the corresponding sub-server and will send sub-server information to the client. The string a main server will send to the client will have the following format: **Test name, Sub-server IP, Sub-server Port**

vi. Using that information, the client will reconnect to the sub server using TCP connection (Note: more than one client can be connected to any particular sub server at a single time). The sub-server will then show different test-types to client. Client will select one and sub-server will assign random marks to client on that test type in the range of 1 to100. The sub server will send the final result to the client and also to the main server for the record of that client. The format of the information will be as follows: Client IP, Client Port, Test Name, Test Type, Marks

vii. The client will show the result on the terminal and then quit. The main server will store the information in a file with the below mentioned format and keep listening for next clients. Client IP, Client Port, Test Name, Test Type, Marks

5. IMPLEMENTATION

a. Main Server

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h> //socket
#include <arpa/inet.h> //inet_addr
#include <stdlib.h>
#include <pthread.h>

/*
 * Main Server
 */

int check_format_of_sub_server_msg(char str[])
{
    int i=0;
    int count=0;
    for(i=0;i<strlen(str);i++)
    {
        if(str[i]==',')
            count++;
    }
    return count;
}

void *sub_server_thread_func (int *client_sock)
{
    printf("starting sub_server_thread_func\n");

    char server_message[2000], client_message[2000];

    //Cleaning the Buffers

    memset(server_message,'\0',sizeof(server_message));
    memset(client_message,'\0',sizeof(client_message));

    //Receive the message from the client

    if (recv(client_sock, client_message, sizeof(client_message),0) < 0)
    {
            printf("sub_server_thread_func Recieve Failed. Error!!!!!\n");
            return;
    }


    printf("\nsub_server_thread_func Client Sock: %i\n",client_sock);
    printf("\nsub_server_thread_func Client Message: %s\n",client_message);
```

```
    if(check_format_of_sub_server_msg(client_message)==2)
    {
        FILE *sub_server_info_File;
        sub_server_info_File = fopen("sub_server_info.txt", "a");
        char entry[200];

        strcpy(entry, client_message);
        strcat(entry, "\n");

        printf("Sub Server info Received: %s",client_message);

        fputs(entry, sub_server_info_File);
        fclose(sub_server_info_File);

    }
    else if(check_format_of_sub_server_msg(client_message)==4)
    {
        FILE *clients_records_File;
        clients_records_File = fopen("clients_records.txt", "a");
        char entry[200];

        strcpy(entry, client_message);
        strcat(entry, "\n");

        printf("Sub Server Message Recieved\nClient Record: %s",client_message);

        fputs(entry, clients_records_File);
        fclose(clients_records_File);
    }
    else
        printf("\nSub_server with socket %i msg format is not
correct!!!\n",client_sock);



    memset(server_message,'\0',sizeof(server_message));
    memset(client_message,'\0',sizeof(client_message));

    //Closing the Socket

    close(client_sock);

    pthread_exit(NULL);
}

int get_sub_server_info(char test_name[],char* info)
{
    memset(info,'\0',sizeof(info));
```

```
    FILE *fp;
    char str[100];

    fp = fopen("sub_server_info.txt", "r");
    if (fp == NULL){
        printf("Could not open file!!\n");
        return 1;
    }
    int flag=0;
    while (fgets(str, sizeof(str), fp) != NULL)
    {
        int i=0;
        flag=1;
        while(i<strlen(test_name))
        {
            if(str[i]!=test_name[i])
            {
                flag=0;
                break;
            }
            i++;
        }
        if(flag==1)
        {
            strcpy(info,str);

            char *newline,*carriage_return;
            newline = strchr(info,'\n');
            if(newline != NULL)
            *newline = '\0';

            carriage_return = strchr(info,'\r');
            if(carriage_return != NULL)
            *carriage_return = '\0';

            return 0;
        }
    }
    fclose(fp);
    return 1;


}


void *client_thread_func (int *client_sock)
{
```

```
    printf("starting client_thread_func\n");
    char server_message[2000], client_message[2000];
    char str[]="Please Enter your Test option\nScience\nMath\nEnglish\n\n";

    //Cleaning the Buffers

    memset(server_message,'\0',sizeof(server_message));
    memset(client_message,'\0',sizeof(client_message));

    strcpy(server_message,str);
    if (send(client_sock, server_message, strlen(server_message),0)<0)
    {
        printf("client_thread_func Send Failed. Error!!!!!\n");
        return;
    }

    //while(1)
    //{
        //Receive the message from the client

        if (recv(client_sock, client_message, sizeof(client_message),0) < 0)
        {
                printf("client_thread_func Receive Failed. Error!!!!!\n");
                return;
        }

        printf("client_thread_func Client Sock: %i\n",client_sock);
        printf("client_thread_func Client Message: %s\n",client_message);

        if(get_sub_server_info(client_message,server_message)==1)
            strcpy(server_message, "Invalid Input!!!");


        //Send the message back to client
        if (send(client_sock, server_message, strlen(server_message),0)<0)
        {
                printf("client_thread_func Send Failed. Error!!!!!\n");
                return;
        }

        memset(server_message,'\0',sizeof(server_message));
        memset(client_message,'\0',sizeof(client_message));


    //}
    //Closing the Socket

    close(client_sock);
```

```
    pthread_exit(NULL);
}


void *sub_server_func (int *sub_socket_desc)
{
    printf("starting sub_server_func\n");
    int client_sock, client_size;
    struct sockaddr_in client_addr;
    char server_message[2000], client_message[2000];
    pthread_t thread4;

    //Cleaning the Buffers

    memset(server_message,'\0',sizeof(server_message));
    memset(client_message,'\0',sizeof(client_message));


    while(1){


        //Accept the incoming Connections

        client_size = sizeof(client_addr);
        client_sock = accept(sub_socket_desc, (struct sockaddr*)&client_addr,
&client_size);

        if (client_sock < 0)
        {
                printf("Accept Failed. Error!!!!!!\n");
                return;
        }
        else
        {

            int ret = pthread_create(&thread4, NULL, sub_server_thread_func,
(int*)client_sock);
            if (ret!=0)
            {
                            printf("Error In Creating Thread\n");
            }
        }

        printf("1-Client Connected with IP: %s and Port No:
%i\n",inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));


    }
    close(sub_socket_desc);
```

```
    pthread_exit(NULL);
}
void *client_func (int *socket_desc)
{
    printf("starting client_func\n");
    int client_sock, client_size;
    struct sockaddr_in client_addr;
    char server_message[2000], client_message[2000];
    pthread_t thread3;

    //Cleaning the Buffers

    memset(server_message,'\0',sizeof(server_message));
    memset(client_message,'\0',sizeof(client_message));


    while(1){


        //Accept the incoming Connections

        client_size = sizeof(client_addr);
        client_sock = accept(socket_desc, (struct sockaddr*)&client_addr,
&client_size);

        if (client_sock < 0)
        {
                printf("Accept Failed. Error!!!!!!\n");
                return;
        }
        else
        {

            int ret = pthread_create(&thread3, NULL, client_thread_func,
(int*)client_sock);
            if (ret!=0)
            {
                            printf("Error In Creating Thread\n");
            }
        }

        printf("1-Client Connected with IP: %s and Port No:
%i\n",inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));


    }
    close(socket_desc);
    pthread_exit(NULL);
}
```

```
int main(int argc, char** argv) {


    int socket_desc, sub_socket_desc, client_sock, client_size;
    struct sockaddr_in server_addr, sub_server_addr, client_addr;
    pthread_t thread1,thread2;

    //Creating Socket

    socket_desc = socket(AF_INET, SOCK_STREAM, 0);
    sub_socket_desc = socket(AF_INET, SOCK_STREAM, 0);

    if(socket_desc < 0)
    {
            printf("Could Not Create Socket. Error!!!!!\n");
            return -1;
    }
    printf("Socket Created\n");
    if(sub_socket_desc < 0)
    {
            printf("Could Not Create Sub Socket. Error!!!!!\n");
            return -1;
    }


    printf("Sub Socket Created\n");

    //Binding IP and Port to socket

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(2000);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    sub_server_addr.sin_family = AF_INET;
    sub_server_addr.sin_port = htons(2001);
    sub_server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    if(bind(socket_desc, (struct sockaddr*)&server_addr, sizeof(server_addr))<0)
    {
            printf("Bind Failed. Error!!!!!\n");
            return -1;
    }

    printf("Bind Done\n");

    if(bind(sub_socket_desc, (struct sockaddr*)&sub_server_addr,
sizeof(sub_server_addr))<0)
    {
```

```
            printf("Sub Bind Failed. Error!!!!!\n");
            return -1;
    }

    printf("Sub Bind Done\n");

    //Put the socket into Listening State

    if(listen(socket_desc, 1) < 0)
    {
            printf("Listening Failed. Error!!!!!\n");
            return -1;
    }

    printf("Listening for Incoming Connections. ... \n");

    if(listen(sub_socket_desc, 1) < 0)
    {
            printf("Sub Listening Failed. Error!!!!!\n");
            return -1;
    }

    printf("Sub Listening for Incoming Connections. ... \n");


    int ret1 = pthread_create(&thread1, NULL, sub_server_func,
(int*)sub_socket_desc);
    if (ret1!=0)
    {
                    printf("Error In Creating Thread1\n");
    }
    int ret2 = pthread_create(&thread2, NULL, client_func, (int*)socket_desc);
    if (ret2!=0)
    {
                    printf("Error In Creating Thread2\n");
    }




    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);
    //Closing the Socket

    //close(client_sock);

    close(socket_desc);
    pthread_exit(NULL); //Terminates the parent thread
```

```
    return (EXIT_SUCCESS);
}
```

b. Sub Server-1 (Science)-

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h> //socket
#include <arpa/inet.h> //inet_addr
#include <stdlib.h>

int send_msg_to_main_server(char msg[])
{

    int socket_desc;
        struct sockaddr_in server_addr;
        char server_message[2000], client_message[2000];

        //Cleaning the Buffers

        memset(server_message,'\0',sizeof(server_message));
        memset(client_message,'\0',sizeof(client_message));

        //Creating Socket

        socket_desc = socket(AF_INET, SOCK_STREAM, 0);

        if(socket_desc < 0)
        {
                printf("Could Not Create Socket. Error!!!!!\n");
                return -1;
        }

        printf("Socket Created\n");

        //Specifying the IP and Port of the server to connect

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(2001);
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

        //Now connecting to the server accept() using connect() from client side

        if(connect(socket_desc, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0)
```

```
        {
                printf("Connection Failed. Error!!!!!");
                return -1;
        }

        printf("Connected\n");

        //Send the message to my information to Main Server

        strcpy(server_message,msg);
        printf("%s",server_message);

        if(send(socket_desc, server_message, strlen(server_message),0) < 0)
        {
                printf("Send Failed. Error!!!!\n");
                return -1;
        }

        memset(server_message,'\0',sizeof(server_message));
        memset(client_message,'\0',sizeof(client_message));

        //Closing the Socket

        close(socket_desc);

        return 0;
}

struct client_info
{
    int socket;
    int port;
    char ip[10];
    struct client_info* loc;
};

void *client_thread_func (void* c_info)
{
    printf("starting client_thread_func\n");

    char server_message[2000], client_message[2000],port_buffer[7];
    struct client_info *info=(struct client_info*) c_info;
    int client_sock=info->socket;
    printf("\nSocket: %i",client_sock);
    printf("\nPort: %i",info->port);
    printf("\nIP: %s",info->ip);
    printf("\nmalloc: %s ",info->loc);

    char test_type[]="Enter your test type:\nPhysics\nChemistry\nBiology\n";
```

```
    //Cleaning the Buffers

    memset(server_message,'\0',sizeof(server_message));
    memset(client_message,'\0',sizeof(client_message));

    strcpy(server_message,test_type);
    if (send(client_sock, server_message, strlen(server_message),0)<0)
    {
        printf("client_thread_func Send Failed. Error!!!!!\n");
        return;
    }

    //Receive the message from the client

        if (recv(client_sock, client_message, sizeof(client_message),0) < 0)
        {
                printf("client_thread_func Receive Failed. Error!!!!!\n");
                return;
        }


        printf("client_thread_func Client Sock: %i\n",client_sock);
        printf("client_thread_func Client Message: %s\n",client_message);

        memset(server_message,'\0',sizeof(server_message));
        memset(port_buffer,'\0',sizeof(port_buffer));
        if(strcmp(client_message,"Biology")==0 ||
strcmp(client_message,"Chemistry")==0 || strcmp(client_message,"Physics")==0)
        {
            sprintf(port_buffer, "%d", info->port);
            strcpy(server_message,info->ip);
            strcat(server_message,",");
            strcat(server_message,port_buffer);
            strcat(server_message,",Science,");
            strcat(server_message,client_message);
            strcat(server_message,",20");
        }
        else
            strcpy(server_message, "Invalid Input!!!");


        //Send the message back to client
        if (send(client_sock, server_message, strlen(server_message),0)<0)
        {
                printf("client_thread_func Send Failed. Error!!!!!\n");
                return;
        }
```

```
        if(send_msg_to_main_server(server_message)==-1)
            return -1;

    close(client_sock);
    free(info->loc);
    pthread_exit(NULL);
}

int Set_connection_for_clients(char my_ip[], char my_port[])
{
    int port=atoi(my_port);


    int socket_desc, client_sock, client_size;
        struct sockaddr_in server_addr, client_addr;
        pthread_t thread;

        //Creating Socket

        socket_desc = socket(AF_INET, SOCK_STREAM, 0);

        if(socket_desc < 0)
        {
                printf("Could Not Create Socket. Error!!!!!\n");
                return -1;
        }

        printf("Socket Created\n");

        //Binding IP and Port to socket

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(port);
        server_addr.sin_addr.s_addr = inet_addr(my_ip);

        if(bind(socket_desc, (struct sockaddr*)&server_addr,
sizeof(server_addr))<0)
        {
                printf("Bind Failed. Error!!!!!\n");
                return -1;
        }

        printf("Bind Done\n");

        //Put the socket into Listening State

        if(listen(socket_desc, 1) < 0)
        {
                printf("Listening Failed. Error!!!!!\n");
```

```
                return -1;
        }

        printf("Listening for Incoming Connections. ... \n");


        while(1)
        {
            //Accept the incoming Connections

            client_size = sizeof(client_addr);
            client_sock = accept(socket_desc, (struct sockaddr*)&client_addr,
&client_size);

            if (client_sock < 0)
            {
                    printf("Accept Failed. Error!!!!!!\n");
                    return -1;
            }
            else
            {
                struct client_info *info;
                info=malloc(sizeof(struct client_info));
                strcpy(info->ip,inet_ntoa(client_addr.sin_addr));
                info->port=ntohs(client_addr.sin_port);
                info->socket=client_sock;
                info->loc=info;

                int ret = pthread_create(&thread, NULL, client_thread_func, (void
*)info);
                if (ret!=0)
                {
                    printf("Error In Creating Thread\n");
                }
            }


            //printf("Client Connected with IP: %s and Port No:
%i\n",inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));
        }

        //Closing the Socket


        close(socket_desc);
        pthread_exit(NULL); //Terminates the parent thread


    return 0;
```

```
}


int main(int argc, char** argv) {

    char msg[100];
    char my_test_name[]="Science";
    char my_ip[]="127.0.0.1";
    char my_port[]="2010";

    memset(msg,'\0',sizeof(msg));

    strcpy(msg,my_test_name);
    strcat(msg,",");
    strcat(msg,my_ip);
    strcat(msg,",");
    strcat(msg,my_port);

    if(send_msg_to_main_server(msg)==-1)
        return -1;

    Set_connection_for_clients(my_ip,my_port);


    return (EXIT_SUCCESS);
}
```

c. Sub Server-2 (Math)-

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h> //socket
#include <arpa/inet.h> //inet_addr
#include <stdlib.h>

int send_msg_to_main_server(char msg[])
{

    int socket_desc;
        struct sockaddr_in server_addr;
        char server_message[2000], client_message[2000];

        //Cleaning the Buffers

        memset(server_message,'\0',sizeof(server_message));
        memset(client_message,'\0',sizeof(client_message));
```

```
        //Creating Socket

        socket_desc = socket(AF_INET, SOCK_STREAM, 0);

        if(socket_desc < 0)
        {
                printf("Could Not Create Socket. Error!!!!!\n");
                return -1;
        }

        printf("Socket Created\n");

        //Specifying the IP and Port of the server to connect

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(2001);
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

        //Now connecting to the server accept() using connect() from client side

        if(connect(socket_desc, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0)
        {
                printf("Connection Failed. Error!!!!!");
                return -1;
        }

        printf("Connected\n");

        //Send the message to my information to Main Server

        strcpy(server_message,msg);
        printf("%s",server_message);

        if(send(socket_desc, server_message, strlen(server_message),0) < 0)
        {
                printf("Send Failed. Error!!!!\n");
                return -1;
        }

        memset(server_message,'\0',sizeof(server_message));
        memset(client_message,'\0',sizeof(client_message));

        //Closing the Socket

        close(socket_desc);

        return 0;
}
```

```
struct client_info
{
    int socket;
    int port;
    char ip[10];
    struct client_info* loc;
};

void *client_thread_func (void* c_info)
{
    printf("starting client_thread_func\n");

    char server_message[2000], client_message[2000],port_buffer[7];
    struct client_info *info=(struct client_info*) c_info;
    int client_sock=info->socket;
    printf("\nSocket: %i",client_sock);
    printf("\nPort: %i",info->port);
    printf("\nIP: %s",info->ip);
    printf("\nmalloc: %s ",info->loc);

    char test_type[]="Enter your test type:\nGeometry\nAlgebra\nIQ\n";

    //Cleaning the Buffers

    memset(server_message,'\0',sizeof(server_message));
    memset(client_message,'\0',sizeof(client_message));

    strcpy(server_message,test_type);
    if (send(client_sock, server_message, strlen(server_message),0)<0)
    {
        printf("client_thread_func Send Failed. Error!!!!!\n");
        return;
    }

    //Receive the message from the client

        if (recv(client_sock, client_message, sizeof(client_message),0) < 0)
        {
                printf("client_thread_func Receive Failed. Error!!!!!\n");
                return;
        }


        printf("client_thread_func Client Sock: %i\n",client_sock);
        printf("client_thread_func Client Message: %s\n",client_message);

        memset(server_message,'\0',sizeof(server_message));
        memset(port_buffer,'\0',sizeof(port_buffer));
```

```
        if(strcmp(client_message,"Geometry")==0 ||
strcmp(client_message,"Algebra")==0 || strcmp(client_message,"IQ")==0)
        {
            sprintf(port_buffer, "%d", info->port);
            strcpy(server_message,info->ip);
            strcat(server_message,",");
            strcat(server_message,port_buffer);
            strcat(server_message,",Math,");
            strcat(server_message,client_message);
            strcat(server_message,",20");
        }
        else
            strcpy(server_message, "Invalid Input!!!");


        //Send the message back to client
        if (send(client_sock, server_message, strlen(server_message),0)<0)
        {
                printf("client_thread_func Send Failed. Error!!!!!\n");
                return;
        }

        if(send_msg_to_main_server(server_message)==-1)
            return -1;

    close(client_sock);
    free(info->loc);
    pthread_exit(NULL);
}

int Set_connection_for_clients(char my_ip[], char my_port[])
{
    int port=atoi(my_port);


    int socket_desc, client_sock, client_size;
        struct sockaddr_in server_addr, client_addr;
        pthread_t thread;

        //Creating Socket

        socket_desc = socket(AF_INET, SOCK_STREAM, 0);

        if(socket_desc < 0)
        {
                printf("Could Not Create Socket. Error!!!!!\n");
                return -1;
        }
```

```
        printf("Socket Created\n");

        //Binding IP and Port to socket

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(port);
        server_addr.sin_addr.s_addr = inet_addr(my_ip);

        if(bind(socket_desc, (struct sockaddr*)&server_addr,
sizeof(server_addr))<0)
        {
                printf("Bind Failed. Error!!!!!\n");
                return -1;
        }

        printf("Bind Done\n");

        //Put the socket into Listening State

        if(listen(socket_desc, 1) < 0)
        {
                printf("Listening Failed. Error!!!!!\n");
                return -1;
        }

        printf("Listening for Incoming Connections. ... \n");


        while(1)
        {
            //Accept the incoming Connections

            client_size = sizeof(client_addr);
            client_sock = accept(socket_desc, (struct sockaddr*)&client_addr,
&client_size);

            if (client_sock < 0)
            {
                    printf("Accept Failed. Error!!!!!!\n");
                    return -1;
            }
            else
            {
                struct client_info *info;
                info=malloc(sizeof(struct client_info));
                strcpy(info->ip,inet_ntoa(client_addr.sin_addr));
                info->port=ntohs(client_addr.sin_port);
                info->socket=client_sock;
                info->loc=info;
```

```
                int ret = pthread_create(&thread, NULL, client_thread_func, (void
*)info);
                if (ret!=0)
                {
                    printf("Error In Creating Thread\n");
                }
            }


            //printf("Client Connected with IP: %s and Port No:
%i\n",inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));
        }

        //Closing the Socket


        close(socket_desc);
        pthread_exit(NULL); //Terminates the parent thread


    return 0;
}


int main(int argc, char** argv) {

    char msg[100];
    char my_test_name[]="Math";
    char my_ip[]="127.0.0.1";
    char my_port[]="2020";

    memset(msg,'\0',sizeof(msg));

    strcpy(msg,my_test_name);
    strcat(msg,",");
    strcat(msg,my_ip);
    strcat(msg,",");
    strcat(msg,my_port);

    if(send_msg_to_main_server(msg)==-1)
        return -1;

    Set_connection_for_clients(my_ip,my_port);


    return (EXIT_SUCCESS);
}
```

d. Sub Server-3 (English) -

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h> //socket
#include <arpa/inet.h> //inet_addr
#include <stdlib.h>

int send_msg_to_main_server(char msg[])
{

    int socket_desc;
        struct sockaddr_in server_addr;
        char server_message[2000], client_message[2000];

        //Cleaning the Buffers

        memset(server_message,'\0',sizeof(server_message));
        memset(client_message,'\0',sizeof(client_message));

        //Creating Socket

        socket_desc = socket(AF_INET, SOCK_STREAM, 0);

        if(socket_desc < 0)
        {
                printf("Could Not Create Socket. Error!!!!!\n");
                return -1;
        }

        printf("Socket Created\n");

        //Specifying the IP and Port of the server to connect

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(2001);
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

        //Now connecting to the server accept() using connect() from client side

        if(connect(socket_desc, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0)
        {
                printf("Connection Failed. Error!!!!!");
                return -1;
        }

        printf("Connected\n");
```

```
        //Send the message to my information to Main Server

        strcpy(server_message,msg);
        printf("%s",server_message);

        if(send(socket_desc, server_message, strlen(server_message),0) < 0)
        {
                printf("Send Failed. Error!!!!\n");
                return -1;
        }

        memset(server_message,'\0',sizeof(server_message));
        memset(client_message,'\0',sizeof(client_message));

        //Closing the Socket

        close(socket_desc);

        return 0;
}

struct client_info
{
    int socket;
    int port;
    char ip[10];
    struct client_info* loc;
};

void *client_thread_func (void* c_info)
{
    printf("starting client_thread_func\n");

    char server_message[2000], client_message[2000],port_buffer[7];
    struct client_info *info=(struct client_info*) c_info;
    int client_sock=info->socket;
    printf("\nSocket: %i",client_sock);
    printf("\nPort: %i",info->port);
    printf("\nIP: %s",info->ip);
    printf("\nmalloc: %s ",info->loc);

    char test_type[]="Enter your test type:\nAnalogies\nAntonyms\nRC
Questions\n";

    //Cleaning the Buffers

    memset(server_message,'\0',sizeof(server_message));
    memset(client_message,'\0',sizeof(client_message));
```

```
    strcpy(server_message,test_type);
    if (send(client_sock, server_message, strlen(server_message),0)<0)
    {
        printf("client_thread_func Send Failed. Error!!!!!\n");
        return;
    }

    //Receive the message from the client

        if (recv(client_sock, client_message, sizeof(client_message),0) < 0)
        {
                printf("client_thread_func Receive Failed. Error!!!!!\n");
                return;
        }


        printf("client_thread_func Client Sock: %i\n",client_sock);
        printf("client_thread_func Client Message: %s\n",client_message);

        memset(server_message,'\0',sizeof(server_message));
        memset(port_buffer,'\0',sizeof(port_buffer));
        if(strcmp(client_message,"Analogies")==0 ||
strcmp(client_message,"Antonyms")==0 || strcmp(client_message,"RC Questions")==0)
        {
            sprintf(port_buffer, "%d", info->port);
            strcpy(server_message,info->ip);
            strcat(server_message,",");
            strcat(server_message,port_buffer);
            strcat(server_message,",English,");
            strcat(server_message,client_message);
            strcat(server_message,",20");
        }
        else
            strcpy(server_message, "Invalid Input!!!");


        //Send the message back to client
        if (send(client_sock, server_message, strlen(server_message),0)<0)
        {
                printf("client_thread_func Send Failed. Error!!!!!\n");
                return;
        }

        if(send_msg_to_main_server(server_message)==-1)
            return -1;

    close(client_sock);
    free(info->loc);
```

```
    pthread_exit(NULL);
}

int Set_connection_for_clients(char my_ip[], char my_port[])
{
    int port=atoi(my_port);


    int socket_desc, client_sock, client_size;
        struct sockaddr_in server_addr, client_addr;
        pthread_t thread;

        //Creating Socket

        socket_desc = socket(AF_INET, SOCK_STREAM, 0);

        if(socket_desc < 0)
        {
                printf("Could Not Create Socket. Error!!!!!\n");
                return -1;
        }

        printf("Socket Created\n");

        //Binding IP and Port to socket

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(port);
        server_addr.sin_addr.s_addr = inet_addr(my_ip);

        if(bind(socket_desc, (struct sockaddr*)&server_addr,
sizeof(server_addr))<0)
        {
                printf("Bind Failed. Error!!!!!\n");
                return -1;
        }

        printf("Bind Done\n");

        //Put the socket into Listening State

        if(listen(socket_desc, 1) < 0)
        {
                printf("Listening Failed. Error!!!!!\n");
                return -1;
        }

        printf("Listening for Incoming Connections. ... \n");
```

```
        while(1)
        {
            //Accept the incoming Connections

            client_size = sizeof(client_addr);
            client_sock = accept(socket_desc, (struct sockaddr*)&client_addr,
&client_size);

            if (client_sock < 0)
            {
                    printf("Accept Failed. Error!!!!!!\n");
                    return -1;
            }
            else
            {
                struct client_info *info;
                info=malloc(sizeof(struct client_info));
                strcpy(info->ip,inet_ntoa(client_addr.sin_addr));
                info->port=ntohs(client_addr.sin_port);
                info->socket=client_sock;
                info->loc=info;

                int ret = pthread_create(&thread, NULL, client_thread_func, (void
*)info);
                if (ret!=0)
                {
                                printf("Error In Creating Thread\n");
                }
            }


            //printf("Client Connected with IP: %s and Port No:
%i\n",inet_ntoa(client_addr.sin_addr),ntohs(client_addr.sin_port));
        }

        //Closing the Socket


        close(socket_desc);
        pthread_exit(NULL); //Terminates the parent thread


    return 0;
}


int main(int argc, char** argv) {

    char msg[100];
```

```
    char my_test_name[]="English";
    char my_ip[]="127.0.0.1";
    char my_port[]="2030";

    memset(msg,'\0',sizeof(msg));

    strcpy(msg,my_test_name);
    strcat(msg,",");
    strcat(msg,my_ip);
    strcat(msg,",");
    strcat(msg,my_port);

    if(send_msg_to_main_server(msg)==-1)
        return -1;

    Set_connection_for_clients(my_ip,my_port);


    return (EXIT_SUCCESS);
}
```

e. Client Server

```
#include <stdio.h>

#include <string.h>
#include <sys/socket.h> //socket
#include <arpa/inet.h> //inet_addr
#include <stdlib.h>

/*
 * client
 */



int main(int argc, char** argv) {



        int socket_desc;
        struct sockaddr_in server_addr;
        char server_message[2000], client_message[2000];

        //Cleaning the Buffers

        memset(server_message,'\0',sizeof(server_message));
        memset(client_message,'\0',sizeof(client_message));
```

```
        //Creating Socket

        socket_desc = socket(AF_INET, SOCK_STREAM, 0);

        if(socket_desc < 0)
        {
                printf("\nCould Not Create Socket. Error!!!!!\n");
                return -1;
        }

        printf("\nSocket Created\n");

        //Specifying the IP and Port of the server to connect

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(2000);
        server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

        //Now connecting to the server accept() using connect() from client side

        if(connect(socket_desc, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0)
        {
                printf("\nConnection Failed. Error!!!!!");
                return -1;
        }

        printf("\nConnected\n");


        //Receive the message back from the server

        if(recv(socket_desc, server_message, sizeof(server_message),0) < 0)
        {
                printf("\nReceive Failed. Error!!!!!\n");
                return -1;
        }

        printf("\nServer Message: \n%s\n",server_message);


        //Get Input from the User

        printf("\nEnter Message: ");
        gets(client_message);

        //Send the message to Server
```

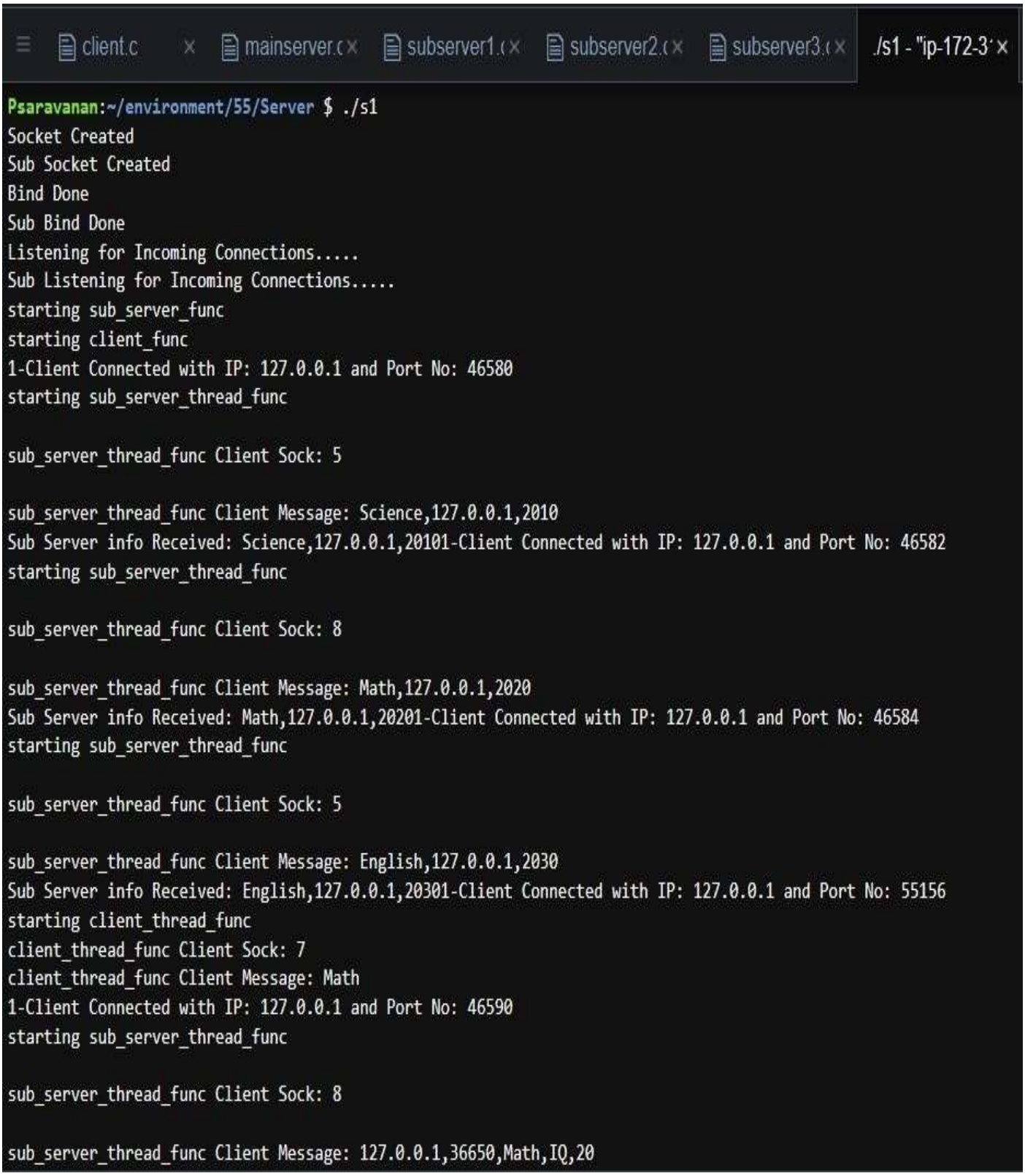
```
        if(send(socket_desc, client_message, strlen(client_message),0) < 0)
        {
                printf("\nSend Failed. Error!!!!\n");
                return -1;
        }

        //Receive the message back from the server
        memset(server_message,'\0',sizeof(server_message));
        if(recv(socket_desc, server_message, sizeof(server_message),0) < 0)
        {
                printf("\nReceive Failed. Error!!!!!\n");
                return -1;
        }


        //strcpy(server_message,"Science,127.0.0.1,2000");
        printf("\nServer info Message: %s\n",server_message);

        char port[10],ip[20],name[10];

        int i=0;
        int j=0;
        for(i=0;server_message[i]!=',';i++)
        {
            name[i]=server_message[i];
        }
        i++;
        for(j=0;server_message[i]!=',';i++,j++)
        {
            ip[j]=server_message[i];
        }
        i++;
        for(j=0;server_message[i]!='\0';i++,j++)
        {
            port[j]=server_message[i];
        }


        int sub_server_port=atoi(port);

        printf("\nName:%s\n",name);
        printf("IP:%s\n",ip);
        printf("port: %i\n",sub_server_port);


        memset(server_message,'\0',sizeof(server_message));
        memset(client_message,'\0',sizeof(client_message));

        //Closing the Socket
```

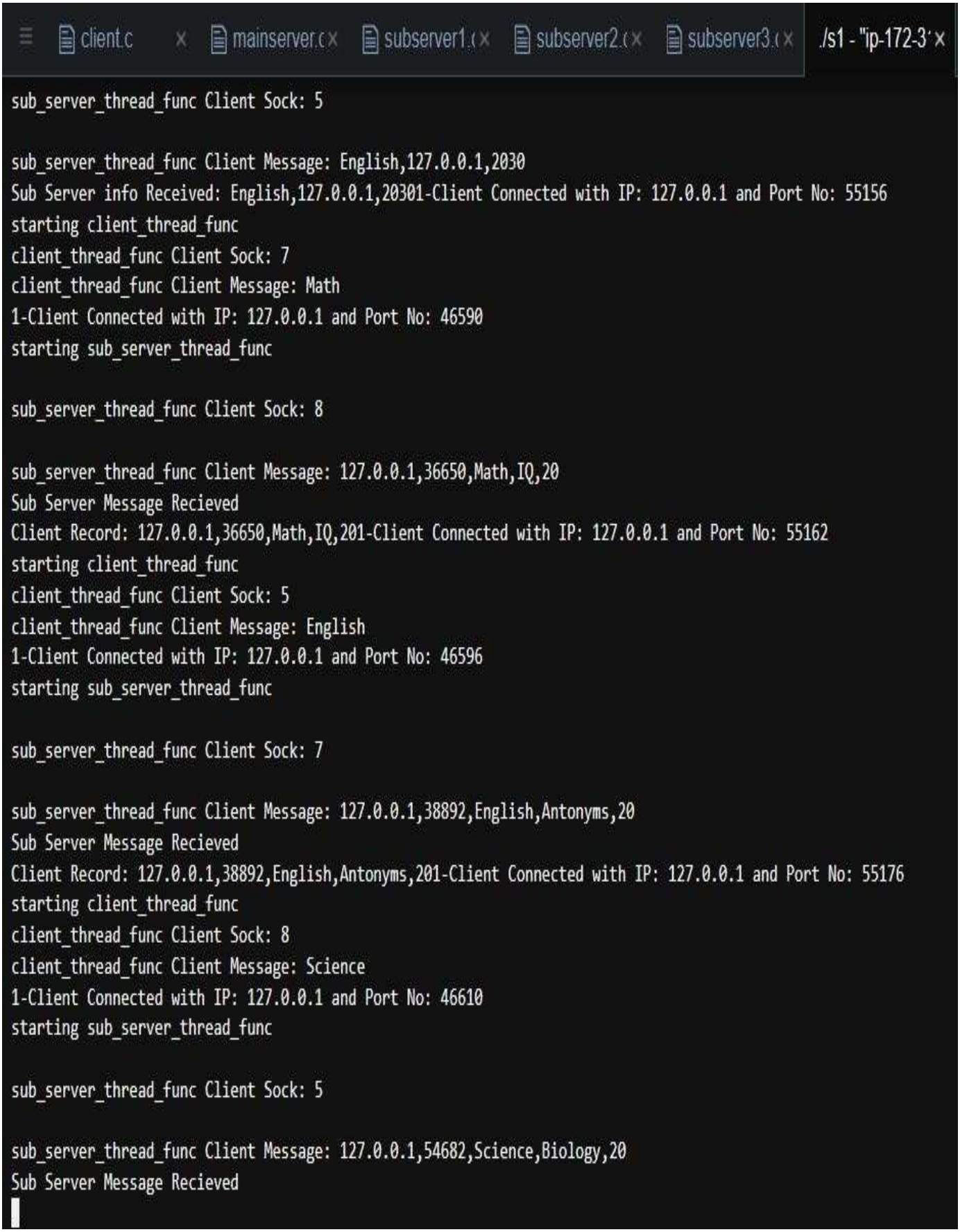
```
        close(socket_desc);

        // Now Connection with Sub-Server

        //Cleaning the Buffers
        //memset(server_message,'\0',sizeof(server_message));
        //memset(client_message,'\0',sizeof(client_message));

        //Creating Socket
        socket_desc = -1;
        socket_desc = socket(AF_INET, SOCK_STREAM, 0);

        if(socket_desc < 0)
        {
                printf("\nCould Not Create Socket. Error!!!!!\n");
                return -1;
        }

        printf("\nSocket Created\n");

        //Specifying the IP and Port of the server to connect

        server_addr.sin_family = AF_INET;
        server_addr.sin_port = htons(sub_server_port);
        server_addr.sin_addr.s_addr = inet_addr(ip);

        //Now connecting to the server accept() using connect() from client side

        if(connect(socket_desc, (struct sockaddr*)&server_addr,
sizeof(server_addr)) < 0)
        {
                printf("\nConnection Failed with Sub Server. Error!!!!!");
                return -1;
        }

        printf("\nConnected to Sub Server\n");

        //Cleaning the Buffers

        memset(server_message,'\0',sizeof(server_message));
        memset(client_message,'\0',sizeof(client_message));

        //Receive the message back from the server

        if(recv(socket_desc, server_message, sizeof(server_message),0) < 0)
        {
                printf("\nReceive Failed. Error!!!!!\n");
                return -1;
```

```
        }

        printf("\nSub Server Message: \n%s\n",server_message);


        //Get Input from the User

        printf("\nEnter Message: ");
        gets(client_message);

        //Send the message to Server

        if(send(socket_desc, client_message, strlen(client_message),0) < 0)
        {
                printf("\nSub Server Send Failed. Error!!!!\n");
                return -1;
        }

        //Receive the message back from the server
        memset(server_message,'\0',sizeof(server_message));
        if(recv(socket_desc, server_message, sizeof(server_message),0) < 0)
        {
                printf("\nSub Server Receive Failed. Error!!!!!\n");
                return -1;
        }


        //strcpy(server_message,"Science,127.0.0.1,2000");
        printf("\nSub Server Message: %s\n",server_message);


        close(socket_desc);



    return (EXIT_SUCCESS);
}
```

6. EXPERIMENT RESULT

a. Main Server




client.c
mainserver.c
subserver1.c
subserver2.c
subserver3.c
./s1 - "ip-172-3"
sub_server_thread_func Client Sock: 5
sub_server_thread_func Client Message: English,127.0.0.1,2030
Sub Server info Received: English,127.0.0.1,20301-Client Connected with IP: 127.0.0.1 and Port No: 55156
starting client_thread_func
client_thread_func Client Sock: 7
client_thread_func Client Message: Math
1-Client Connected with IP: 127.0.0.1 and Port No: 46590
starting sub_server_thread_func
sub_server_thread_func Client Sock: 8
sub_server_thread_func Client Message: 127.0.0.1,36650,Math,IQ,20
Sub Server Message Recieved
Client Record: 127.0.0.1,36650,Math,IQ,201-Client Connected with IP: 127.0.0.1 and Port No: 55162
starting client_thread_func
client_thread_func Client Sock: 5
client_thread_func Client Message: English
1-Client Connected with IP: 127.0.0.1 and Port No: 46596
starting sub_server_thread_func
sub_server_thread_func Client Sock: 7
sub_server_thread_func Client Message: 127.0.0.1,38892,English,Antonyms,20
Sub Server Message Recieved
Client Record: 127.0.0.1,38892,English,Antonyms,201-Client Connected with IP: 127.0.0.1 and Port No: 55176
starting client_thread_func
client_thread_func Client Sock: 8
client_thread_func Client Message: Science
1-Client Connected with IP: 127.0.0.1 and Port No: 46610
starting sub_server_thread_func
sub_server_thread_func Client Sock: 5
sub_server_thread_func Client Message: 127.0.0.1,54682,Science,Biology,20
Sub Server Message Recieved

b. Sub-Server 1 (Science)

```
Psaravanan:~/environment/55/Server $ ./sub1
Socket Created
Connected
Science,127.0.0.1,2010Socket Created
Bind Done
Listening for Incoming Connections.....
starting client_thread_func

Socket: 4
Port: 54682
IP: 127.0.0.1
malloc:  client_thread_func Client Sock: 4
client_thread_func Client Message: Biology
Socket Created
Connected
```

c. Sub-Server 2 (Math)

```
Psaravanan:~/environment/55/Server $ ./sub2
Socket Created
Connected
Math,127.0.0.1,2020Socket Created
Bind Done
Listening for Incoming Connections.....
starting client_thread_func

Socket: 4
Port: 36650
IP: 127.0.0.1
malloc:  client_thread_func Client Sock: 4
client_thread_func Client Message: IQ
Socket Created
Connected
```

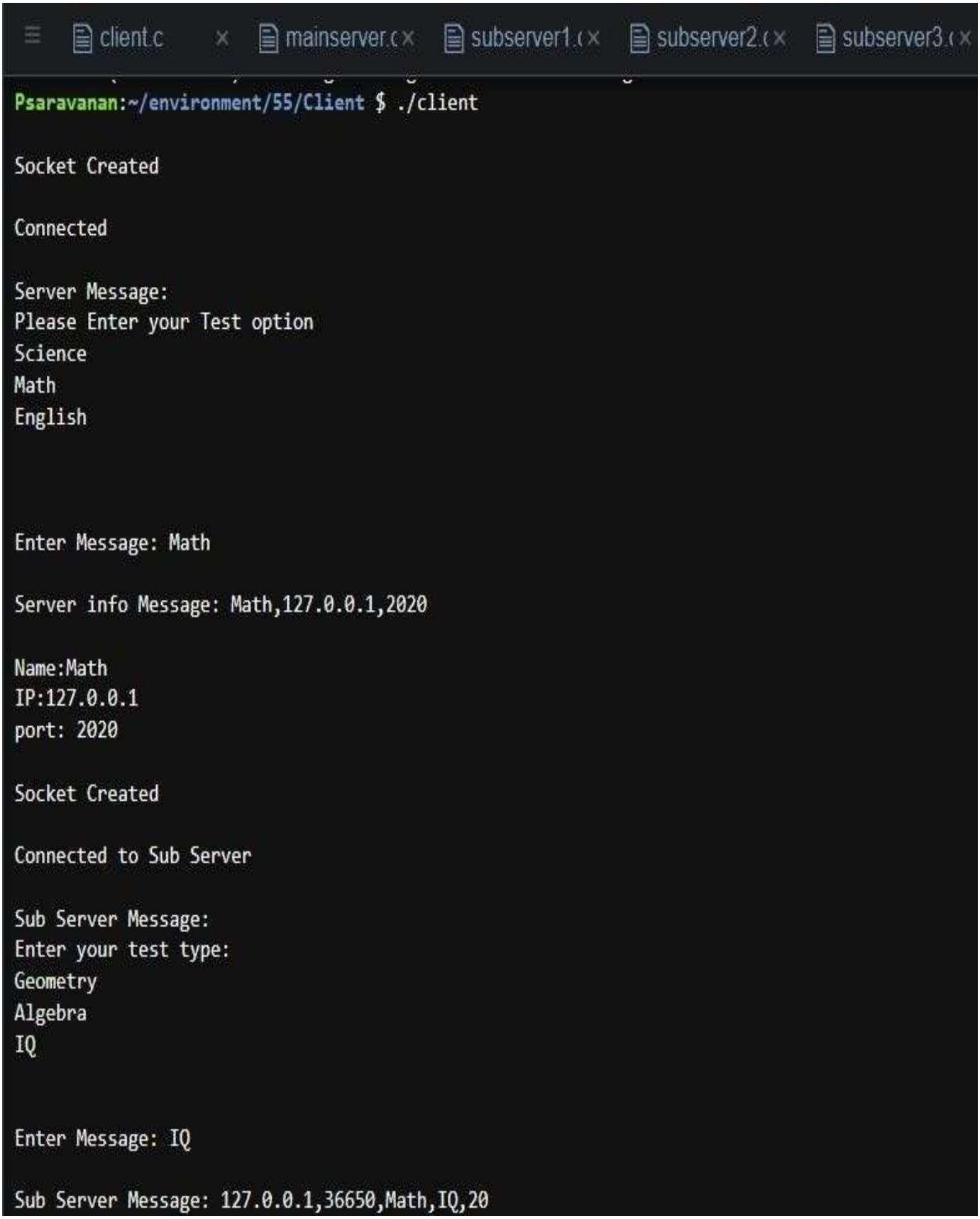
d. Sub-Server 3 (English)

```
Psaravanan:~/environment/55/Server $ ./sub3
Socket Created
Connected
English,127.0.0.1,2030Socket Created
Bind Done
Listening for Incoming Connections.....
starting client_thread_func

Socket: 4
Port: 38892
IP: 127.0.0.1
malloc:  client_thread_func Client Sock: 4
client_thread_func Client Message: Antonyms
Socket Created
Connected
```

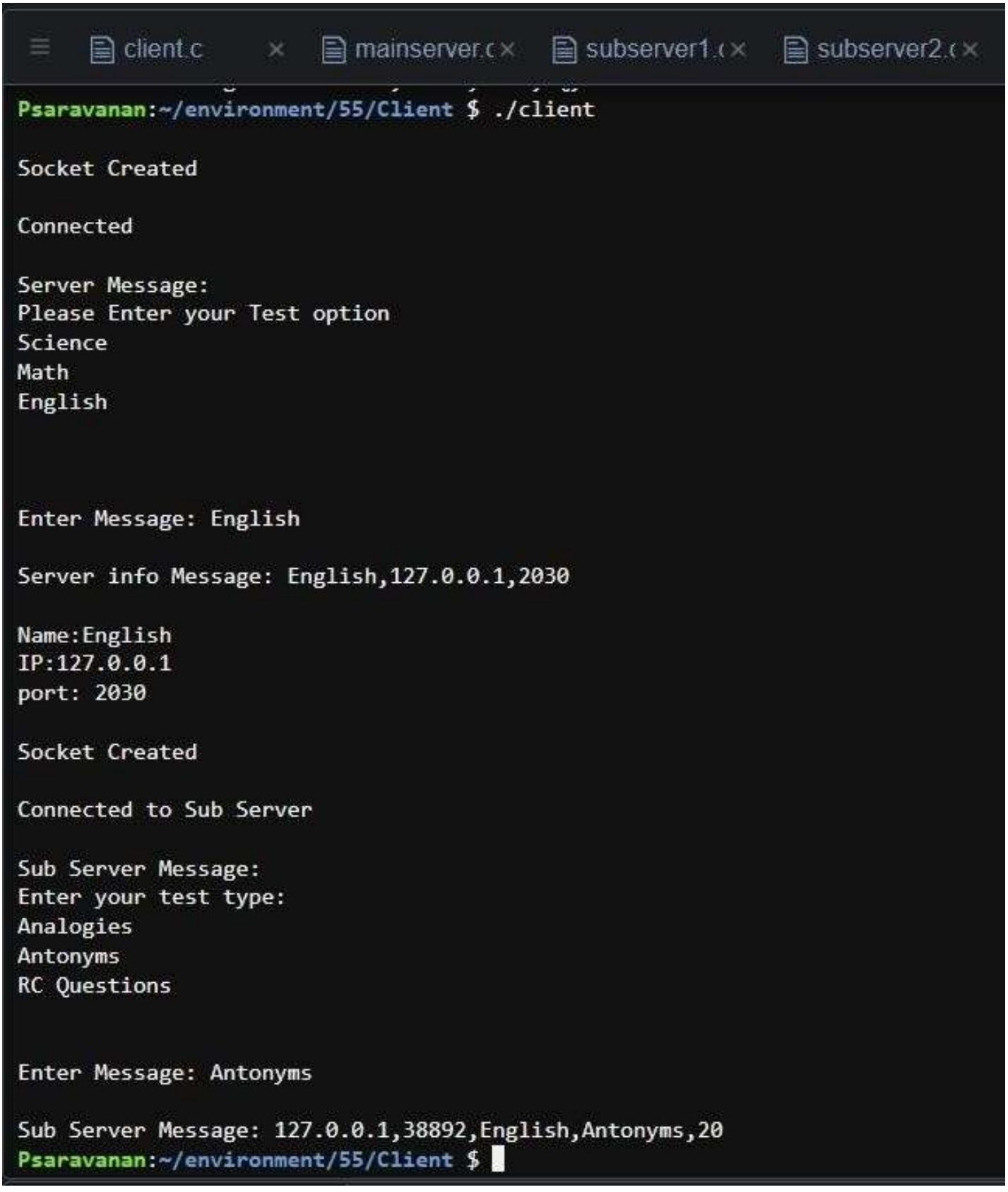
e. Client Terminal 1 (Science)

```
Psaravanan:~/environment/55/Client $ ./client

Socket Created

Connected

Server Message:
Please Enter your Test option
Science
Math
English



Enter Message: Science

Server info Message: Science,127.0.0.1,2010

Name:Science
IP:127.0.0.1
port: 2010

Socket Created

Connected to Sub Server

Sub Server Message:
Enter your test type:
Physics
Chemistry
Biology


Enter Message: Biology

Sub Server Message: 127.0.0.1,54682,Science,Biology,20
Psaravanan:~/environment/55/Client $
```

f. Client Terminal 2 (Math)



g. Client Terminal 3 (English)



7. RESULT ANALYSIS AND CONCLUSION

As depicted through the above screenshots our model creates 3 subserves for each of the subjects and can be effectively used to conduct quizzes as well as exams and can be further expanded for as many subjects as required. We have used TCP server connections to perform the task. The package was designed in such a way that future modifications can be done easily. The following conclusions can be deduced from the development of the project:

i. Automation of the entire system improves the efficiency
ii. It provides a friendly graphical user interface which proves to be better when compared to the existing system.
iii. It gives appropriate access to the authorized users depending on their permissions.
iv. It effectively overcomes the delay in communications.
v. Updating of information becomes so easier.
vi. System security, data security and reliability are the striking features.

From these points we can conclude that this could be an efficient system for online examination modes and as a result of its simplicity, can easily be built further on in multiple ways.

8. REFERENCES

a. https://www.tutorialspoint.com

b. https://www.javatpoint.com

c. https://www.w3schools.com