Project Title:

Multithreaded-Online-Quiz-System

Submitted by:

Qurat Ul Ain	**2021-CE-02**
Noor Fatima	**2021-CE-07**
Ansa Aslam	**2021-CE-21**

Submitted to:

Mam Darakshah

Course:

Computer Networks

Semester:

6th

Date of Submission:

7th April, 2024

Department of Computer Engineering

University of Engineering and Technology, Lahore

TABLE OF CONTENTS:

Introduction

The Online Quiz System is designed to facilitate multiple clients simultaneously taking tests in various subjects such as Science, Mathematics, and English. This system employs TCP socket programming for communication between the main server, sub-servers, and clients. The architecture includes a main server and three sub-servers, each responsible for a specific type of test. Clients connect to the main server, select a test type, and are redirected to the corresponding sub-server to complete their test.

Methodology

The working methodology of an Online Quiz System involves several key steps and processes, from client interaction to test administration and result handling. Let's delve into each aspect in detail:

1. System Architecture

The Online Quiz System typically consists of the following components:

- Main Server: Responsible for handling client connections, managing sub-server information, and routing client requests to appropriate sub-servers based on selected test types.

- Sub-Servers: Specialized servers dedicated to administering specific types of tests (e.g., Science, Mathematics, English).

- Clients: Users accessing the system to take tests, interact with the main server, and connect to designated sub-servers to complete assessments.

2. Client Interaction

a. Client Connection:

1. Client Initialization: The client initiates a connection to the main server using TCP/IP protocols.

2. Main Server Response: Upon successful connection, the main server presents a list of available test types (e.g., Science, Mathematics, English) to the client.

b. Test Selection:

1. Client Test Selection: The client chooses a test type (e.g., Mathematics) based on the options provided by the main server.

2. Main Server Routing: Using its database of sub-server information, the main server determines which sub-server is responsible for administering the selected test type.

3. Client Redirect: The main server provides the client with the IP address and port of the designated sub-server responsible for the chosen test type.

3. Sub-Server Interaction

a. Sub-Server Initialization:

1. Sub-Server Registration: Each sub-server registers with the main server upon startup, providing information about the test type it supports (e.g., Mathematics).

b. Client Connection:

1. Client Connection to Sub-Server: The client establishes a new connection to the designated sub-server using the provided IP address and port.

c. Test Administration:

1. Test Presentation: The sub-server presents the client with test options corresponding to the selected test type (e.g., Geometry, Algebra, IQ for Mathematics).

2. Client Test Interaction: The client interacts with the sub-server to complete the test, selecting answers or providing responses.

d. Result Generation:

1. Score Calculation: Upon completion of the test, the sub-server calculates a random score (between 1 and 100) for the client's performance.

2. Result Communication: The sub-server sends the test results (including client IP, port, test type, and score) back to both the client and the main server for logging and recording.

4. Result Handling

a. Client Feedback:

1. Result Display: The client receives and displays the test score and feedback from the sub-server.

b. Logging and Record-Keeping:

1. Main Server Logging: The main server records the client's test results (including client details, test type, and score) in a database or file for administrative purposes.

5. Client Termination

a. Connection Closure:

1. Client Disconnection: After receiving the test results, the client terminates the connection with the sub-server.

b. System Interaction:

1. System Reset: The main server and sub-servers remain operational, ready to handle new client connections and test requests.

6. Error Handling and Security

The Online Quiz System should incorporate robust error handling mechanisms and security measures:

- Error Detection: Implement checks for connection errors, timeouts, or unexpected behavior to ensure reliable system performance.

- Data Validation: Validate incoming data from clients and sub-servers to prevent unauthorized access or malicious activities.

- Encryption: Use encryption techniques (e.g., SSL/TLS) for secure communication between clients, main server, and sub-servers to protect sensitive data.

7. Scalability and Performance

Design the Online Quiz System to be scalable and performant:

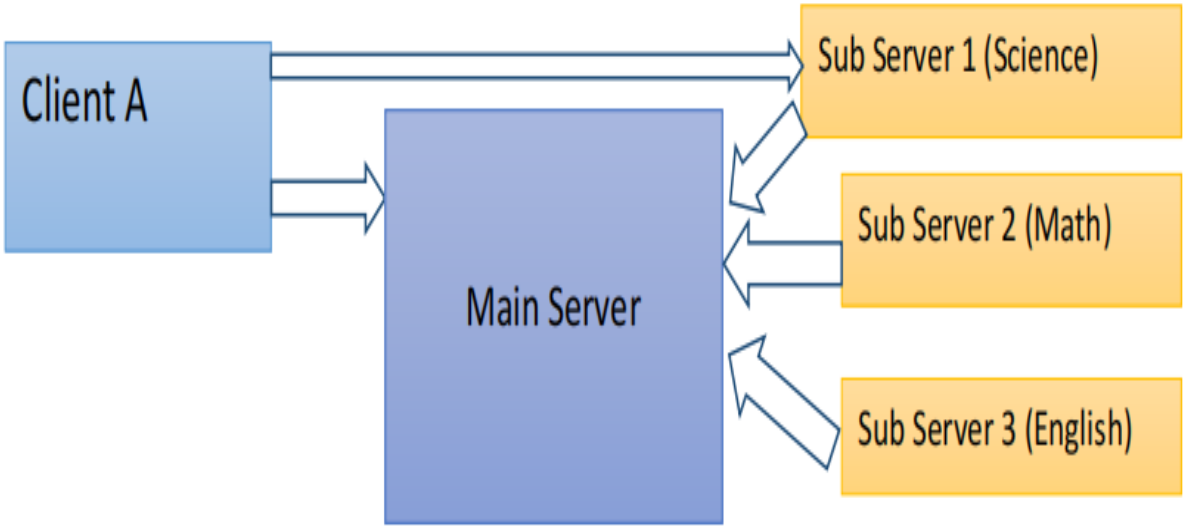
- Concurrency: Handle multiple client connections and test sessions concurrently to support a large number of users.

- Resource Management: Optimize resource usage (CPU, memory, network bandwidth) to ensure smooth system operation under varying loads.

Solution Approach

The solution approach for implementing an Online Quiz System involves designing a system architecture with distinct components: a Main Server, Sub-Servers (dedicated to Science, Mathematics, and English tests), and Clients. Communication is facilitated using TCP/IP socket programming, ensuring reliable data exchange. A database is employed to store sub-server information, client details, and test results. The Main Server listens for connections, manages sub-server information, and routes client requests to the appropriate sub-server based on selected test types. Sub-Servers connect to the main server, register supported test types, administer tests, calculate random scores, and communicate results back to clients and the main server. Clients interact with the main server to select test types, connect to designated sub-servers, complete tests, and view scores. Error handling, data validation, and encryption (e.g., SSL/TLS) are implemented for security. The system is designed for scalability, supporting concurrent users, and optimized resource management for performance. Test results are logged in a database or file for administrative purposes and analysis, ensuring a seamless and secure Online Quiz System experience.

Flow Chart



Features

This section outlines the key features and functionalities of the Online Quiz System:

- **Scalability:** Discuss the system's ability to handle multiple clients and sub-servers concurrently.
- **Flexibility:** Highlight how clients can choose from different test types (Science, Mathematics, English) based on their preferences.
- **Centralized Control:** Emphasize the centralized management of test assignments and result collection by the main server.
- **Error Handling:** Mention mechanisms implemented for error detection and recovery using TCP protocols.
- **Data Logging:** Describe how client test results are recorded and logged for administrative purposes.

Functional requirements

Detail the functional requirements necessary for the successful implementation of the Online Quiz System:

- **Main Server Requirements:** List the capabilities required for the main server, such as handling client connections, managing sub-server information, and routing client requests.
- **Sub-Server Requirements:** Specify the functionalities expected from each sub-server, including registering with the main server, administering tests, and reporting results.
- **Client Requirements:** Outline the actions and interactions expected from clients, such as connecting to the main server, selecting test types, and receiving test results.

Working the program

This section provides a step-by-step explanation of how the Online Quiz System operates:

- **Main Server Operation:** Describe how the main server listens for connections, handles client requests, and communicates with sub-servers.
- **Sub-Server Functionality:** Explain the role of each sub-server in administering tests, generating random scores, and communicating results.
- **Client Interaction:** Detail the sequence of actions performed by clients, including connecting to the main server, selecting tests, and interacting with sub-servers to complete assessments.
- **Result Handling:** Describe how test results are communicated between sub-servers, clients, and the main server for recording and logging.

Applications

Highlight the potential applications and use cases of the Online Quiz System:

- **Educational Institutions**: Explain how the system can be used for conducting online assessments and quizzes in educational settings.
- **Training Centers:** Discuss the system's relevance in evaluating candidates remotely for training and certification programs.
- **Online Platforms:** Describe how the system can be deployed on online platforms offering skill assessment services to users.
- **Corporate Training:** Mention the utility of the system for companies conducting employee evaluations and training exercises.

Code Implementation

Main-server.c

```c
/*
 * File:   Main_Server.c
 * Author: Muneeb Ahmad
 */
#include <stdio.h>
#include <string.h>
#include <sys/socket.h> //socket
#include <arpa/inet.h> //inet_addr
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
```

```c
/*
 * Main Server
 */
```

```c
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 0;
   }

   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 0;
}

//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 0;
    }

    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 0;
    }
```

```
    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 0;
    }
    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 0;
        }
        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);
}
```

Sub-server-1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h> //socket
#include <arpa/inet.h> //inet_addr
#include <unistd.h>
#include <pthread.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 0;
}

//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 0;
```

```
    }
```

```
    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 0;
    }
```

```
        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);
}
```

Sub-server-2

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h> //socket
#include <arpa/inet.h> //inet_addr
#include <stdlib.h>
#include <unistd.h>
#include <pthread.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 0;
}

//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 0;
    }
```

```
    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 0;
        }


        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);
}
```

Sub-server-3

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h> //socket
#include <arpa/inet.h> //inet_addr
#include <stdlib.h>
#include <unistd.h>
#include <pthread.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 0;
}

//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 0;
   }
```

```
   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 0;
    }

    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);
}
```

Client.c

```
/*
```

```
* File:   client.c
* Author: Muneeb Ahmad
*
*/
```

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

Softwares Used

- **Programming Language:** C (for server and client implementation).
- **TCP Socket Programming Libraries:** socket module in C.
- **Text Editor/IDE:** Any preferred Python IDE (e.g., PyCharm, VS Code).
- **Operating System:** Any OS supporting Python (e.g., Windows, Linux, macOS).

Conclusion

In summary, the Online Quiz System developed using TCP Multithreading technology efficiently manages quizzes across diverse subject areas. By utilizing a main server and specialized sub-servers for Science, Mathematics, and English tests, the system handles concurrent client connections effectively. TCP/IP communication ensures reliable data transfer, while random test scoring adds dynamism to the assessment process. This project showcases practical implementation of networking and distributed system design, with potential for future enhancements in test features, security, and user experience.

References

1. C Documentation: https://docs.c.org/3/library/socket.html

2. TCP/IP Sockets in C: https://realc.com/python-sockets/

3. Computer Networking: A Top-Down Approach by James F. Kurose and Keith W. Ross (for networking concepts).