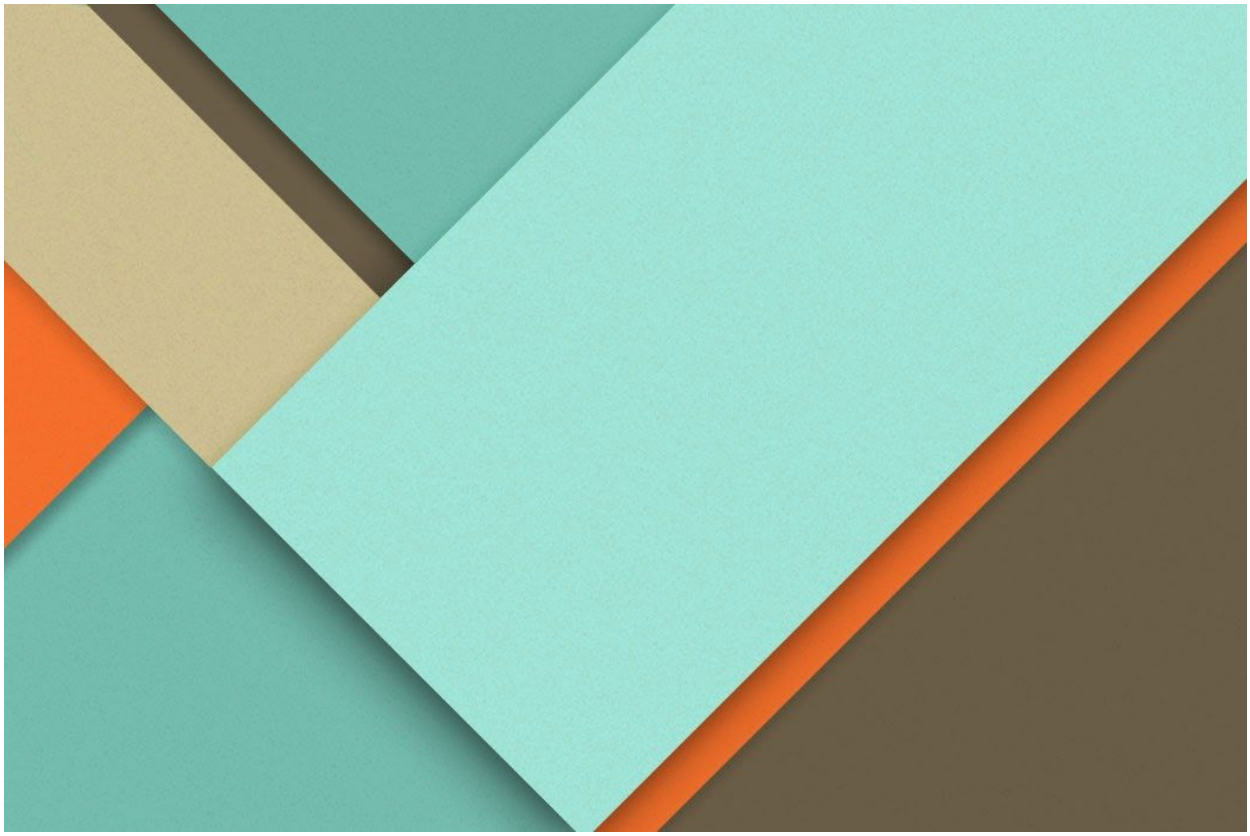

Trivia Game Application using socket programming in C

Group - 5

Overview

Develop a game , where multiuser participate in the game. Server has 5 question with each question four options. if users connected to the server, server start sending the question one by one with a timestamp reply of 1 minute. If multiuser playing the game, then winner decided by the server based on the average minimum time taken a client to reply all answers. If client gives a wrong reply to anyone of the answers, then server send a message to client "better luck next time" and terminate the connection of the client.

Ngrok and SSH

Ngrok is a very lightweight tool that creates a secure tunnel on your local machine along with a public URL you can use for browsing your local site.

We use TCP protocol for connecting different hosts.

The below command is used by the host.

```
./ngrok <protocol> <port-number>
```

- **./ngrok:** The executable that starts the ngrok service.
- **protocol:** The protocol that needs to be used. Here, we use the **tcp** protocol.
- **port-number:** The port to be used for the connection. Here, we used the port **22.**

The below command is used by the client.

```
ssh <host-user-name>@<ip-address> -p<port-number>
```

- **ssh:** It provides a secure encrypted connection between two devices over an insecure network.
- **host-user-name:** The username of the host we are trying to connect.
- **ip-address:** The IP address given by the ngrok at the host side. It is of the pattern “0.tcp.ngrok.io” or “2.tcp.ngrok.io”.
- **port-number:** The virtual port number used to establish the connection. It is a 5-digit number.

socket

It creates a socket. It returns a socket description, similar to a file handler.

```
int socket(domain, type, protocol);
```

- **domain:** communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)
- **type:** communication type
  ➔ SOCK_STREAM: TCP(reliable, connection oriented)
  ➔ SOCK_DGRAM: UDP(unreliable, connectionless)
- **protocol:** communication protocol. Protocol value for Internet Protocol(IP), which is 0.

sockaddr_in

Structure describing an Internet socket address.

```
struct sockaddr_in{
   __SOCKADDR_COMMON (sin_);
```

```
    in_port_t sin_port;
    struct in_addr sin_addr;


    unsigned char sin_zero[sizeof (struct sockaddr) -
                __SOCKADDR_COMMON_SIZE -
                sizeof (in_port_t) -
                sizeof (struct in_addr)];
};
```

- **sin_family:** It is the communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)
- **sin_port:** It stores the port number for connection.
- **sin_addr:** It stores the IP address for connection. Here, INADDR_ANY is used.

bind

It binds the socket to the address and port number specified in addr(custom data structure).

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t
addrlen);
```

- **sockfd:** It stores the socket description.
- **addr:** It is the structure that stores the address and port number to which the socket must bind to.
- **addrlen:** It is the size of the aforementioned structure.

listen

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection.

```
int listen(int sockfd, int backlog);
```

- **sockfd:** It stores the socket description.
- **backlog:** It is the maximum number of clients that can wait in the queue for connection.

accept

It extracts the first client connection in the queue.

```
int new_socket= accept(int sockfd, struct sockaddr *addr,
socklen_t *addrlen);
```

- **new_socket:** It stores the socket description of the incoming client.
- **sockfd:** It stores the server socket description.
- **addr:** It is the structure that stores the address and port number to which the socket must accept.
- **addrlen:** It is the size of the aforementioned structure.

connect

It sends a connection request to the server.

```
int connect(int sockfd, struct sockaddr *addr, socklen_t
addrlen);
```

- **sockfd:** It stores the socket description.
- **addr:** It is the structure that stores the address and port number of the client.
- **addrlen:** It is the size of the aforementioned structure.

send

It sends a message to another socket.

```
int send(int sockfd, void *message, int size, int flags);
```

- **sockfd:** It stores the socket description.
- **message:** It is a pointer to the message that needs to be received. It can be any datatype, or a custom structure.
- **size:** It is the size of the buffer.
- **flags:** It is used to send any flags.

recv

It receives a message from another socket.

```
int recv(int sockfd, void *message, int size, int flags);
```

- **sockfd:** It stores the socket description.
- **message:** It is a pointer to the message that needs to be received. It can be any datatype, or a custom structure.
- **size:** It is the size of the buffer.
- **flags:** It is used to receive any flags from the sender.

pollfd

Data structure describing a polling request.

poll

Poll the file descriptors described by the NFDS structures starting at FDS. If TIMEOUT is nonzero and not -1, allow TIMEOUT milliseconds for an event to occur; if TIMEOUT is -1, block until an event occurs.Returns the number of file descriptors with events, zero if timed out,or -1 for errors.

```
int poll(struct pollfd* fds, nfds_t nfds, int timeout);
```

- **pollfd:** It stores the file descriptors. Here, we use STDIN.
- **nfds:** It sores the number of file descriptors. Here, we use 1.

- **timeout:** It is the time until which the forthcoming instructions are executed in milliseconds. Here, we use 60000 ms.

time_t

typedef long time_t

Returned by time.

time

This function is defined in the time.h header file. This function returns the time since 00:00:00 UTC, January 1, 1970 in seconds.

```
time_t time(time_t* timer);
```

- **timer:** It is a pointer to the timer_t variable which will store the current time

ctime

This function is defined in the time.h header file. This function function returns the string representing the localtime based on the argument timer in human readable format.

```
char *ctime(time_t *timer);
```

- **timer**: This is the pointer to a time_t object that contains a calendar time.

sleep

Sleep function delays program execution for a given number of seconds.

```
int sleep(int seconds);
```

- **seconds:** Time the process has to wait in seconds.

pthread.h

The library used to create a multithreaded application is pthread.h

The data types used are

- pthread_t
- pthread_mutex_t

The functions used are

- pthread_create
- pthread_detach
- pthread_exit
- pthread_mutex_lock
- pthread_mutex_unlock

pthread_t

It is the data type used to uniquely identify a thread.

pthread_mutex_t

It is the data type of a mutex object.

pthread_create

It is used to create a new thread.

```
int pthread_create(pthread_t * thread, const pthread_attr_t *
attr, void * (*start_routine)(void *), void *arg);
```

- **thread:** pointer to an unsigned integer value that returns the thread id of the thread created.
- **attr:** pointer to a structure that is used to define thread attributes. Set to NULL for default thread attributes.
- **start_routine:** It is a pointer to a subroutine that is executed by the thread. The return type and parameter type of the subroutine must be of type void *.
- **arg:** It is a pointer to void that contains the arguments to the function defined in the earlier argument

pthread_detach

It is used to detach a thread. A detached thread does not require a thread to join on terminating. The resources of the thread are automatically released after terminating if the thread is detached.

```
int pthread_detach(pthread_t thread);
```

- **thread:** The thread ID which needs to be detached.

pthread_exit

It is used to terminate a thread.

```
int pthread_detach(pthread_t thread);
```

- **thread:** The thread ID which needs to be detached.

Custom structures used

We are using a separate structure for keeping the note of the client i.e. the detail about the client name, port number, their ip address, the total time client takes to solve the questions, whether the client has completed all the questions, also whether the client has qualified or not.

Also we have structure to store all the questions and the options along with the correct answer to each of the questions.

```
//Structure for info of client
struct info{
   char name[30];
   int clientSocket;
   struct sockaddr_in clientAddr;
   float totalTime;
   bool done;
   bool qualified;
};
```

```
//Structure for questions and answer
struct QA{
   char question[1024];
   char choice[4][30];
   int answer;
};
```

Since we are using the pthread which takes only one argument. Hence, we created a structure of the required variables to be passed to the function, which was passed to the function in pthread_create.

Compiling the program

Programs with the pthread.h library cannot be compiled directly by the gcc command.

```
gcc <file-name.c> -o <output-file-name> -pthread
```

- **gcc:** The C-compiler in unix systems.
- **file-name.c:** The file that needs to be compiled.
- **output-file-name:** The executable output file name.
- **-pthread:** This tells the compiler to use the pthread library to compile.