QuizzGame

Autor: Alexandru Covalciuc

Universitatea “Alexandru Ioan Cuza”, Iasi , 700506, Romania

Abstract:

QuizzGame este o aplicatie interactiva prin care un utilizator isi poate testa cunostintele de cultura generala. Aplicatia are la baza un server **TCP/IP** care coordoneaza si gestioneaza un numar nelimitat de clienti folosind **thread-uri**. Comunicarea server client se realizeaza folosind **socket-uri.**

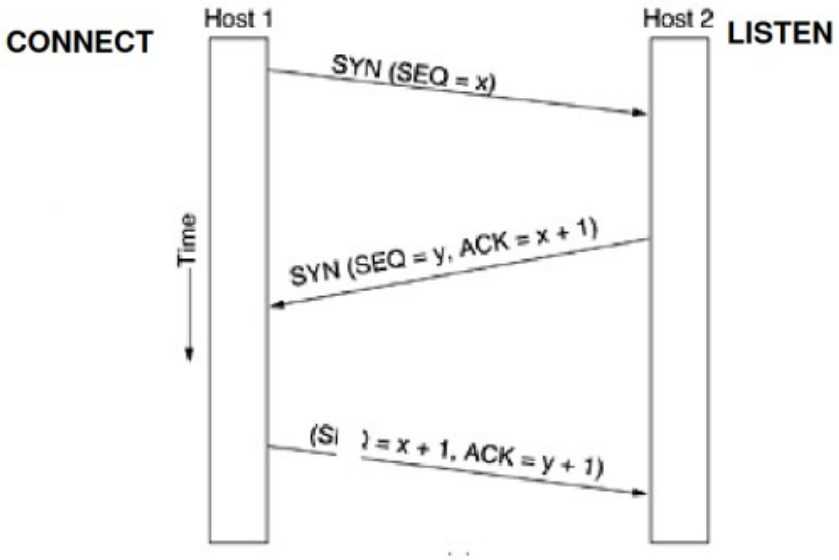
1 Tehnologii utilizate

In acest capitol prezentam pe scurt tehnologiile si tehnicile implementate in cadrul proiectului.

1.1 Server TCP

Serverul implementeaza protocolul TCP, deoarece acesta este un protocol de comunicare care asigura o comunicare secventiala si fara pierdere de informatie.

Serverul stabileste noi conexiuni aplicand un “tree way handshake”, ilustrat in figura de mai jos.



1.2 Gestinonare clienti folosind thread-uri

Thread-urile reprezinta subprocese care executa un bloc de instructiuni. In cadrul QuizzGame, serverul atribuie un thread fiecarui client care stabileste o conexiune . Metodele pentru gestionarea thread-urilor se regasesc in libraria <pthread.h>

1.3 Comunicare prin socket-uri

Socket-urile sunt interfete I/O bidirectionale prin intermediul carora serverul TCP comunica cu clientii. Metodele pentru gestionarea socket-urilor se regasesc in librariile <sys/types.h> si <sys/socket.h>.

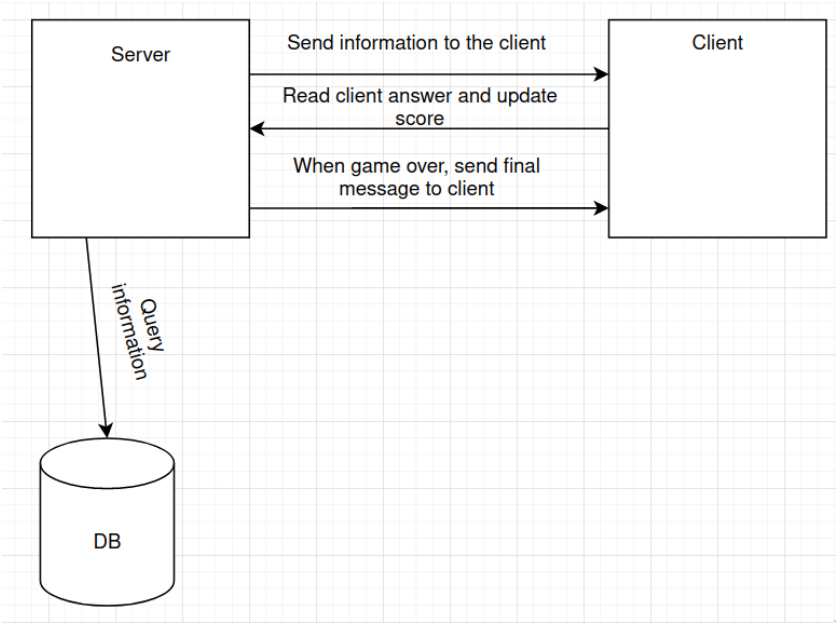
1.4 Multiplexare

Serverul introduce functionalitatea de “timeout” prin monitorizarea socket-ului cu ajutorul caruia se realizeaza comunicarea cu clientii. Procedeul este realizat folosind functia select(), inclusa in libraria <sys/select.h>

1.5 Mutex and shared variables

Procesele de tip thread modifica variable aflate in zode comune de memorie. Sincronizarea accesului la aceste variabile se realizeaza folosind mutex-uri.

2 Arhitectura aplicatiei



3 Detalii de implementare

- Creere si initializare socket

```
/* crearea unui socket */
if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
  {
    perror ("[server]Eroare la socket().\n");
    return errno;
  }
/* utilizarea optiunii SO_REUSEADDR */
int on=1;
setsockopt(sd,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on));
```

- Ascultare dupa clienti

```
while (1)
  {

    if ( (client = accept (sd, (struct sockaddr *) &from, &length)) < 0)
  {
    perror ("[server]Eroare la accept().\n");
    continue;
  }
```

- Gestionarea de noi conexiuni prin atribuirea creerea de thread-uri

```
td=(struct thData*)malloc(sizeof(struct thData));
td->idThread=i++;
td->cl=client;

    /*adaugam socketul in set*/
pthread_create(&th[i], NULL, &treat, td);
```

- Serverul extrage si formateaza in mod secvential informatia din baza de date. Conexiunea se realizeaza folosind un handle de tipul sqlite3*, iar informatia este extrasa prin executarea prepared statement de tipul sqlite3_stmt*. Dupa inceperea executiei, fiecare thread deschide o conexiune catre baza de date, si genereaza un nou prepared statement, distribuind pe rand intrebarile catre client.

```
//initializam conexiunea la baza de date
sqlite3* database_handle = init_sql_db_connection(db_path);
sqlite3_stmt* prepared_statement = create_prepared_statement(database_handle,query);
```

- Incarcarea de intrebari din baza de date se face folosind functia “extrage_rand_formatat_baza_de_date”. Aceasta extrage informatia din prepared_statement-ul oferit ca argument, si formateaza fiecare coloana a rezultatului.

```
stop = sqlite3_step(prepared_statement);
if (stop == SQLITE_ROW)
{

    for(int i=0;i<rows_no;i++)
    {
        switch(i)
        {
            case 0:
                sprintf(data,"%s) ", sqlite3_column_text(prepared_statement,i));
                break;
            case 1:
                sprintf(data+strlen(data),"%s\n", sqlite3_column_text(prepared_statement,i));
                break;
            case 6:
                sprintf(data+strlen(data),"\n%s ", sqlite3_column_text(prepared_statement,i));
                break;
            default:
                //pentru a evita o eroare de compilare
                printf("");
                //reinitializam varianta de raspuns
                char varianta[255]="";
                // Generam litera din fata raspunului. Variantele de raspuns incep de la coloana 2,
                // deci trebuie sa adaptam valoarea variabilei i.
                varianta[0] = 'A' + i -2;
                varianta[1] = '.';
                strcat(varianta+2, sqlite3_column_text(prepared_statement,i));
                sprintf(data+strlen(data),"%s ", varianta);
                break;
        }
```

Serverul limiteaza timpul pe care un client il are la dispozitie pentru a returna un raspuns. Prin intermediul functiei select serverul verifica daca au fost inregistrate operatii pe socket-ul indicat, iar in functie de codul returnat actualizeaza scorul total al clientului (daca timpul a expirat serverul nu contorizeaza punctajul ultimului raspuns).

```
//reinitializam timpul maxim permis
tv.tv_sec=5;
tv.tv_usec=0;
//reinitializam lista de fd
FD_ZERO(&file_descriptors_set);
FD_SET(tdL.cl,&file_descriptors_set);
errno = select(tdL.cl+1,&file_descriptors_set,NULL,NULL,&tv);
if (errno == -1)
{
    printf("Eroare la select");
    break;
}
```

- Terminarea jocului are loc atunci cand un client parcurge toate intrebarile venite de la server. Inainte de finalizarea executiei thread-ul corespunzator clientului castigator actualizeaza variabila “shared” stop_condition si trimite catre client mesajul de instiintare cu privire la incheierea jocului.
La fiecare iteratie a jocului thread-urile verifica daca valoarea variabilei stop_condition s-a modificat. In cazul variabila este schimbata, thread-ul trimite catre client un mesaj prin care anunta finalizarea jocului si isi incheie executia.
Sincronizarea accesului la variabila “shared” este realizata pe baza de mutex-uri.

```
pthread_mutex_lock(&winner_mutex);


if(stop_condition !=0)
{

sprintf(mesaj_final,"Jocul s-a incheiat! Castigator: Jucatorul %d .\n Ati obtinut %d puncte.\n Felicitari!",winner_id,score);
    /* transmitem scorul clientului */
if(errno !=-1){
if (write (tdL.cl,mesaj_final, strlen(mesaj_final)+1) <= 0)
    {
        printf("[Thread %d] deconectat.\n",tdL.idThread);
        errno=-1;
    }
}

}

pthread_mutex_unlock(&winner_mutex);
```

Bibliografie

1. Compute Networks - UAIC (2022), https://profs.info.uaic.ro/~computernetworks/
2. Andrew S. Tanenbaum et al., Computer Networks (2010).
3. SQLite Documentation, https://www.sqlite.org/docs.html
4. Linux Man pages documentation, https://linux.die.net/man/