QuizzGame

Glodeanu Irina-Elena

Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iaşi

[irina\_glodeanu96@info.uaic.ro](mailto:irina_glodeanu96@info.uaic.ro)

**Abstract.** Aplicatia functioneaza precum un joc intre trei clienti pe parcursul caruia acestora li se pun o serie de cinci intrebari cu cate patru variante de raspuns. La sfarsitul acestui joc se va afisa pentru fiecare atat cate puncte a obtinut cat si catigatorul global. Se trateaza situatia in care un utilizator s-ar deconecta in mijlocul jocului astfel incat partida sa continue intre utilizatorii ramasi. Se pot desfasura un numar variabil de jocuri in paralel.

**Key words:** server multithreading, TCP/IP, thread, socket, baza de date SQLite

**1.Introducere**

Aceasta este o aplicatie client/server care sa ofere functionalitatea unui joc de quiz in care se pot conecta un numar nelimitat de clienti care pot juca 3 cate 3 in cate o sesiune.

Serverul va coordona clientii care raspund la un set de intrebari pe rand, in ordinea in care s-au inregistrat. Comunicarea intre server si client se va realiza folosind socket-uri. Toata logica va fi realizata in server, clientul doar raspunde la intrebari.

Serverul sincronizeaza toti clienti intre ei si ofera fiecaruia un timp de n secunde pentru a raspunde la cate o intrebare dupa care verifica raspunsul dat de client si daca este corect va retine punctajul pentru acel client.

Se vor pune cate x intrebari, fiecare cu 4 variante de raspuns. (in aplicatie am fixat x=3 pentru a putea demonstra functionalitatea acesteia in timp util)

Vor fi folosite baze de date SQLite atat pentru stocarea intrebarilor si a raspunsurilor, cat si a informatiilor despre utilizatori.

Pentru cazul in care un participant paraseste jocul, serverul va gestiona situatiile in astfel incat jocul sa continue fara probleme pentru ceilalti .

**2. Tehnologiile utilizate**

1. **TCP**

Pentru aceasta aplicatie folosesc un server TCP concurrent (engl. Transmission Control Protocol , Protocolul de control al transmisiei).

Acest protocol realizeaza o conexiune viruala full-duplex intre client si server intre care se face schimb de informatie(flux de octeti).

Am considerat ca acest protocol este mai potrivit pentru aplicatia in cauza deoarece comunicarea se face constant intre client si server deopotriva( TCP-ul fiind un protocol orientat conexiune). De asemenea, avem nevoie de fiabilitatea data de acest protocol deoarece pentru o buna functionare a aplicatiei, informatiile transmise intre cilent si server nu pot fi pierdute sub nicio forma (orice pierdere poate duce a nefunctionarea aplicatiei). Din acelasi motiv, folosim TCP-ul deoarece ne este asigurata livrarea pachetelor de date in ordinea in care au fost transmise. In plus, fiind un protocol folosit de obicei de aplicații care au nevoie de confirmare de primire a datelor, error checking-ul, message acknoledgement-ul (receprorul raspunde cu un mesaj de confirmare) si retransmiterea in caz de octeti pierduti (caracteristici pe care UDP-ul, spre exemplu, nu le are), consider ca acest protocol reprezinta cea mai potrivita alegere din punctul de vedere al functionalitatii necesare aplicatiei mele.

In cazul detectiei de probleme precum pierderea, duplicarea sau livrarea in alta ordine decat cea initiala a datelor(datorita congestiilor de retea, incarcarea traficului, etc.), TCP-ul solicita retransmisia pachetelor pierdute, rearanjează pachetele în ordine si ajută la minimizarea traficul din rețea în vederea reducerii apariției altor probleme.

Desi TCP-ul nu are ca principal scop livrarea foarte rapida a datelor (pentru aceasta ar fi trebuit sa folosim UDP), ci mai degraba a livrarii lor exacte, acest lucru nu ne deranjeaza, deoarece , in cazul aplicatitei in cauza, transmisia sigura a datelor este prioritara.

Am utilizat un model concurent deoarece pentru a indeplini scopul unui joc intre mai multi utilizatori cat si de a realiza mai multe partide concomitent, trebuie sa avem capacitatea de a deservi mai multi clienti in acelasi timp.

1. **SQLite3**

SQLite este o librarie care implementeaza un motor de baze de date SQL.

Am folosit aceasta tehnologie deoarece

* SQLite nu are nevoie de un server separat sau sistem pentru a opera.(serverless)
* SQLite nu are nevoie de configuratie (nu are nevoie de setup or administratie)
* O baza de date SQLite completa poate fi salvata in simple fisiere de pe disk.
* SQLite nu are dependente externe.(self-contained)
* SQLite permite acces sigur pentru procese/thread-uri .
* SQLite e rapid si nu necesita setup, ceea ce face testarea usoara.

**3.Arhitectura aplicatiei (conceptele implicate, diagrama aplicatiei)**

Aplicatia va folosi o arhitectura de tipul CLIENT/SERVER TCP concurent.

Serverul va primi cererile de conectare de la clienti, moment in care acestia au posibilitatea de a-si alege profilul : jucatori sau admini.

Un admin va trebui sa cunoasca o parola prestabilita (in cazul acestui proiect – “321”) pentru a avea acces la anumite comenzi ce se pot efectua asupra bazelor de date (quiz.db, useri.db) - stergeri, update-uri, fetch-uri.

Jucatorilor le va fi oferita posibilitatea de se loga( in cazul in care clientii detin un cont si o parola potrivita acelui cont) sau de a se inregistra in caz contrar. Daca numele clientului sau parola nu corespund cu un user autorizat sa acceseze aplicatia, acestuia i se da ocazia sa repete actiunea.

Odata cu verificarea de catre server existentei acelui user, clientul este pus sa astepte pana se logheaza si alti clienti pana se formaza o grupare de trei, moment in care se da drumul a joc.

Jocul se structureaza sub forma a 3(sau alt numar de) intrebari cu cate 4 variante de raspuns pentru care fiecare jucator, in parte, are timp 5 secunde sa raspunda cu indexul unei variante. Fiecare jucator primeste intrebarea pe care a primit-o si jucatorul anterior abia dupa ce raspunde acesta.

La finalul celor trei intrebari, fiecare jucator va fi anuntat, in parte, cate puncte a strans (fiecare intrebare raspunsa corect=1 punct) si castigatorul/castigatorii jocului.

Dupa acest verdict clientii sunt inchisi iar serverul nu, acesta asteptand alti clienti sa se conecteze sau gestionand alt(e) joc(uri).

****

**figura: diagrama aplicatiei**

**4.Detalii de implementare (cod relevant, comentat; *use-case*-uri)**

Dupa activarea serverului si mai apoi a conectarii concurente a clientilor la portul aferent, acestia au posibilitatea logarii (fie ca admini, fie ca jucatori).

Server

**Bucla infinita din int main() care ne ajuta sa acceptam oricati clienti**

int j=1;

while(1) //bucla infinita cu ajutorul careia acceptam un numar nelimitat de clienti

{

.

.

.

pthread\_mutex\_lock(&mlock);

if ((client = accept (sd, (struct sockaddr \*) &from, &length)) < 0)

{

perror ("[thread]Eroare la accept().\n");

}

pthread\_mutex\_unlock(&mlock);

threadsPool=(struct Thread\*)realloc(threadsPool, j\*sizeof(struct Thread)); //alocam blocuri de memorie ori de cate ori se conecteaza un cient

j++;

threadsPool[nr\_cl].cl=client; //punem in threadsPool[nr\_cl].cl descriptorul returnat de accept

threadsPool[nr\_cl].idTh=i; //threadsPool[nr\_cl].idTh-id-ul threadului la care suntem

i++;

flag\_meniu=0;

while(flag\_meniu==0) //pentru a deservi clientului in a-si alege tipul : admin sau jucator normal

{

if(login(threadsPool[nr\_cl].cl, threadsPool[nr\_cl].idTh)==1) //daca este jucator normal

{

nr\_cl++; //incrementam numarul de clienti

flag\_meniu=1;

srand(time(NULL)); //pentru modul random de a pune intrebarile

pthread\_create(&fir\_executie, NULL, &grupare, NULL); //cream un thread pt gruparea clientilor cate 3 pt a juca

}

else //daca e admin

{

... //realizam toate actiunile in primul thread

}

**Structura functiei login(..)**

int login(int cl,int idThread)

{

...

while(flag\_sau==0) //daca variabila "sau" nu este fie 0 fie 1 atunci citeste din nou de la client

{

if(read (cl, &sau,sizeof(sau))<= 0)

{

...

}

if(sau==0) //daca e jucator

{

...

return 1;

} else if(sau==1) // daca clientul e admin - in main merg pe ramura else..

{

....

return 0;

}

**Functia grupare() in care grupez jucatorii 3 cate 3 si asigur deservirea lor in ordinea conectarii**

static void \*grupare()

{

int i;

pthread\_t joc;

camera \*camera\_jucatori; //aceasta va fi structura ("camera") in care se va tine jocul

int array[3];

while (1)

{

inregistrare(8," "," ",0,1); //selecteaza ID-urile utilizatorilor logati

inregistrare(9," "," ",0,1); //verifica cati sunt logati si pune aceasta valoare in variabila "disponibili"

if (disponibili % 3==0) //din 3 in 3 facem un joc

{

inregistrare(10," "," ",0,1); //punem in array[0]primul utilizator LOGAT dpdv al id-lui in tabela

array[0] = ar0;

inregistrare(10," "," ",0,1); //punem in array[0] al doilea utilizator LOGAT dpdv al id-lui in tabela

array[1] = ar0;

inregistrare(10," "," ",0,1); //punem in array[0] al treilea utilizator LOGAT dpdv al id-lui in tabela

array[2] = ar0;

pp=0;

camera\_jucatori = (struct camera\*)malloc(sizeof(struct camera)); //alocam memorie pt structura camera

int t=0; //cand se face 3 (se conecteaza cei 3 jucatori) se pun intr-un thread nu sa joace

char user[20]=" ";

for (i = m; i < nr\_cl; i++) //facem urmatoarele actiuni pt grupari de 3 clienti pana cand terminam clentii

{

if (threadsPool[i].id == array[0])

{

inregistrare(18," "," ",0,threadsPool[i].id); //selecteaza numele utilizatorului de la threadsPool[i].id si il punem in variabila 'user\_pt\_joaca'

inregistrare(4,user\_pt\_joaca," ",0,threadsPool[i].id); //updatam campul JOACA=1 pentru acest user

if(i%3==0)camera\_jucatori->cl1=threadsPool[i].cl; //realizam transmiterea intrebarilor in ordinea conectarii

if(i%3==1)camera\_jucatori->cl2=threadsPool[i].cl;

if(i%3==2)camera\_jucatori->cl3=threadsPool[i].cl;

t++;

}

if (threadsPool[i].id == array[1]) //idem mai sus

{

inregistrare(18," "," ",0,threadsPool[i].id);

inregistrare(4,user\_pt\_joaca," ",0,threadsPool[i].id);

if(i%3==0)camera\_jucatori->cl1=threadsPool[i].cl;

if(i%3==1)camera\_jucatori->cl2=threadsPool[i].cl;

if(i%3==2)camera\_jucatori->cl3=threadsPool[i].cl;

t++;

}

if (threadsPool[i].id == array[2]) //idem mai sus

{

inregistrare(18," "," ",0,threadsPool[i].id);

inregistrare(4,user\_pt\_joaca," ",0,threadsPool[i].id);

if(i%3==0)camera\_jucatori->cl1=threadsPool[i].cl;

if(i%3==1)camera\_jucatori->cl2=threadsPool[i].cl;

if(i%3==2)camera\_jucatori->cl3=threadsPool[i].cl;

t++;

}

if (t == 3) //se va ajunge aici abia cand se vor fi conectat 3 clienti iar in

{ //camera\_jucatori->cl1camera\_jucatori->cl2 si camera\_jucatori->cl3

//voi avea descriptorii returnati de accept pt fiecare client din joc

m=m+3; //serverul va fi pregatit sa faca acelasi mecanism si pt urmatorii 3 conectati si tot asa

printf("Intra serverul in camera jocuri \n");

pthread\_create(&joc, NULL, &quiz, camera\_jucatori);

while(oku);

oku=0;

break;

}

}

}

sleep(10);

}

return(NULL);

}

**Functia quiz() in care realizez comunicarea cu fiecare utilizator in parte , respectand si toate cerintele de sincronizare, in vederea jucarii unei partide de quiz**

static void \*quiz(void \*arg)

{

struct camera aux;

aux = \*((struct camera\*)arg); //aici am cei 3 descriptori pentru cei 3 clienti din sesiune

.

.

.

//modul in care intrebarile sunt puse random : punem id-urile lor intr-un vector cu ajutorul functiei rand()

while( y < nr\_intrebari\_de\_pus ) // nr\_intrebari\_de\_pus e definita cu define ca 3

{

random[y]=rand() % max\_id + 1; //aleg randm id-ul intrebarilor pe care le pun

for(z=0; z<y && z!=y ;z++)

if(random[z]==random[y]) //am grija sa nu se repete. daca se repeta aleg din nou pt pozitia aia y--;

y++;

}

.

.

.

inregistrare(11,user\_nou\_1," ",5,1); //delogam clientul cand a terminat de jucat (updatam logat cu 0 )

inregistrare(11,user\_nou\_2," ",5,1);

inregistrare(11,user\_nou\_3," ",5,1);

inregistrare(19,user\_nou\_1," ",5,1); // nu mai joaca (updatam JOACA cu 0)

inregistrare(19,user\_nou\_2," ",5,1);

inregistrare(19,user\_nou\_3," ",5,1);

}

**Client**

**#define FLUSH\_STDIN(x) {if(x[strlen(x)-1]!='\n'){do fgets(Junk,16,stdin);while(Junk[strlen(Junk)-1]!='\n');}else x[strlen(x)-1]='\0';}**

**char Junk[16]; // buffer-ul pentru aruncarea imputului excesiv**

**// folosint pt "FLUSH\_STDIN"**

**...**

**int main (int argc, char \*argv[])**

**{**

**signal(SIGINT, INThandler); //daca clientul va apasa din acest punct ctrl-C,**

**//acesta functia signal va duce clientul in fuctia**

**//INThandler, unde SIGINT e semnalul pt ctrl-C**

**...**

//aici multe verificari si etape ale autentificarii/inregistrarii astfel incat toate cazurile sunt acoperite

......

char gunoi;

int flags = fcntl(STDIN\_FILENO, F\_GETFL, 0);

fcntl(STDIN\_FILENO, F\_SETFL, flags | O\_NONBLOCK);

while(read( STDIN\_FILENO, &gunoi, sizeof(gunoi))!=-1 );

fcntl(STDIN\_FILENO, F\_SETFL, flags);

//STDIN\_FILENO - default standard input file descriptor number which is 0

//F\_GETFL - specifica pt fcntl faptul ca ar trebui returnate flagurile asociate cu STDIN\_FILENO

//F\_SETFL - seteaza descriptorii de fisiere asociati cu STDIN\_FILENO. pentru asta e nevoie de campul flags

// structura care ne da posibilitatea de a da unui “concurent” un anumit timp de raspuns

ret = poll(&fd, 1, 5000); // 5 secunde pt input (sa se intample evenimentul fd.events = POLLIN )

while(flag\_gata\_raspunsul==0) //daca flag\_gata\_raspunsul=0 inseamna ca inca se mai poate raspunde

{ // flag\_gata\_raspunsul devine 1 fie atunci cand exista un raspuns de trimis serverului

switch (ret) //fie ca ai raspuns corect sau gresit (sau nu ai raspuns ceea ce inseamna ca ai gresit)

{

case -1: //eroare la poll

break;

case 0: //am iesit din timp

if(flag\_ctrlc==0)

{

flag\_gata\_raspunsul=1;

printf("Ai iesit din timp. Ai 0 puncte pt intrebarea asta\n\n");

flag\_999=1; //trimite un raspuns gresit la server astfel incat sa nu se potriveasca cu varianta corecta

} //nu se primeste niciun punct

break;

default:

if(flag\_ctrlc==0) //daca clientul n-a apasat ctrl-c ....

{

if(read(0,&rasp2,sizeof(rasp2))<0) //..se citeste de la input raspunsul dat de jucator

{

perror ("[client]Eroare la read() de la server.\n");

return errno;

}

rasp=atoi(rasp2);

printf("Ai raspuns %i. Orice mai scrii e igonrat.\n",rasp);

if(rasp==1||rasp==2||rasp==3||rasp==4) flag\_gata\_raspunsul=1;

else if(rasp==5) //clientul a apasat deconectare

{

question= nr\_intrebari\_de\_pus + 2; // ca sa ies din bula mare ((while<=nr\_intrebari\_de\_pus)) si acel utilizator sa nu mai primeasca intrebari

flag\_gata\_raspunsul=1;

flag\_deconect=1;

}

else

{ printf("Introdu 1,2,3,4! \n");

printf("Raspunsul este(1,2,3 sau 4) (5 pt deconectare): ");

}

}

break;

}

}

flag\_gata\_raspunsul=0; // sa pot raspunde la urmatoarea intrebare

if(flag\_999==1) // daca s-a iesit din timp la intrebarea curenta

{

rasp=999;

if(write(sd,&rasp,sizeof(rasp))<=0) //trimitem un raspuns gresit la server (fie 999)

{

perror ("[client]Eroare la write() spre server.\n");

return errno;

}

}else if (write(sd,&rasp,sizeof(rasp))<= 0) //daca nu s-a iest din timp trimitem raspunsul care s-a dat la server

{

perror ("[client]Eroare la write() spre server.\n");

return errno;

}

.......

}

void INThandler(int sig)

{

char c;

signal(sig, SIG\_IGN); //primeste signalul (s-a apasat ctrl-c)

printf(" -> Ati apasat ctrl-c, rasp se va transforma in 5 si veti fi deconectat.\n");

flag\_gata\_raspunsul=1;

flag\_ctrlc=1; // o sa facem flag\_ctrlc la else-uri ca sa putem da write de oriunde

rasp=5;

question= nr\_intrebari\_de\_pus + 2;

signal(SIGINT, INThandler); // cand iese din functie se continua programul in mod normal

// dar cu modificarile facute

}

A se da o importanta deosebita functiei **pthread\_create (pthread\_t \**thread*, const pthread\_attr\_t \**attr*, void \*(\**start\_routine*)(void\*), void \**arg*);** folosita atat pentru gruparea jucatorilor pt joc, cat si pentru cautarea acelui numar necesar de jucatori.

De asemenea functia **poll** asteapta pentru ca un eveniment sa se intample pe un file descriptor

**int poll(struct pollfd \****fds***, nfds\_t** *nfds***, int** *timeout***);**

- nfds =numarul de filedescriptors

- *timeout =cate secunde asteptam sa se intample eventul (la noi e input)*

Baze de date:

* **useri.bd**
* baza de date in care sunt stocate conturile si informatiile despre utilizatori
* contine campurile :
  + ID INT PRIMARY KEY
    - id-ul userului
  + USER CHAR(50)
    - numele/contul userului
  + PAROLA CHAR(50)
    - parola aferenta
  + LOGAT INT
    - 0-utilizator disponibil (nelogat)
    - 1-utilizator logat
  + REZ INT
    - rezultatul sau in joc
  + JOACA INT
    - 0- utilizator disponibil (nu joaca)
    - 1-utilizator care joaca deja
* **quiz.bd**
* baza de date in care sunt stocate intrebarile ce vor fi mai apoi incarcate in aplicatie
* contine campurile :
  + ID
    - id-ul intrebarii in tabela
  + INTREBARE
  + RASP1
  + RASP2
  + RASP3
  + RASP4
  + CORECT
    - varianta corecta de raspuns

Folderul **libr** contine headerele si codul de configurare pentru lucrul cu baze de date SQLite. (de pe site-ul https://www.sqlite.org/download.html )



figura: usecases

**5.Concluzii (e.g. cum ar putea fi imbunatatita solutia propusa? )**

Utilitare care se mai pot aduce (implementa) aplicatiei pot fi:

-posibilitatea introducerii unui numar variabil de jucatori de catre primul jucator care se conecteaza

-posibilitatea adaugarii unui meniu pentru admin din care sa manipuleze jocul/bazele de date

-interfata grafica (mai) interactiva

**6.Bibliografie**

<http://students.info.uaic.ro/~adria/teach/courses/net/cursullaboratorul.php>

Informatiile acumulate in timpul cursurilor si laboratorului de “Retele de calculatoare”

<https://sqlite.org/>