Online Info Olympiad

Lipan Radu-Matei A1

Facultatea de Informatica Iasi, Iasi, Romania

1 Introducere

Proiectul "Online Info Olympiad" presupune implementarea unui model client-server prin intermediul caruia sa se poata tine o olimpiada de informatica online. Fiecare concurent se va conecta cu ajutorul aplicatiei client la server pe baza unor date de autentificare. Dupa autentificare vor primi o serie de probleme de rezolvat in limbajul C++ si vor fi anuntati de cat timp au la dispozitie pentru rezolvarea acestora in cadrul concursului. Cand acestia au terminat problemele sau cand s-a terminat timpul pus la dispozitie pentru acestea, sursele vor fi trimise serverului pentru a fi notate automat. Dupa ce toti participantii au fost notati, clasamentul va fi trimis tuturor clientilor de catre server. Serverul va contine fisiere de configurare ce vor descrie numarul maxim de concurenti, problemele selectate si modul de selectie al acestora, timpul pus la dispozitie etc.

2 Tehnologiile utilizate

Pentru transmiterea mesajelor dintre client si server in acest proiect am folosit protocolul TCP.

Deoarece viteza de transmitere a mesajelor nu este critica, insa corectitudinea mesajelor si ordinea lor este critica, folosirea protocolul UDP nu aduce niciun avantaj in cadrul acestui proiect. Mai mult, protocolul UDP ar putea afecta integritatea concursului, posibila pierdere sau modificare a unui mesaj in cadrul transmiterii informatiei unui fisier sursa ar putea afecta in mod nedrept punctajul unui concurent. Faptul ca protocolul TCP este orientat conexiune, fara pierdere de informatii si cu control al erorilor asigura integritatea concursului din punctul de vedere prezentat mai sus.

3 Arhitectura aplicatiei

3.1 Concepte implicate

Atat clientul cat si serverul vor implementa multiplexarea canalelor de intrare prin intermediul primetivei select(). Clientul va primi comenzi atat de la server cat si de la tastatura, iar serverul va primi comenzi de la toti clientii sai. De fiecare data cand este primita o noua comanda, aceasta este detectata de select si este executata imediat.

In cazul serverului, primirea datelor, abordarea si transmiterea raspunsului la un mesaj primit de la un client se va face in cadrul unui nou thread. Cate un thread pentru fiecare client care a trimis un mesaj indentificat de primitiva select este lansat in executie. Acest thread se va ocupa de rezolvarea comenzii respective a clientului dupa care va fi inchis. Prin intermediul multiplexarii si a folosirii thread-urilor pentru rezolvarea comenzilor clientilor se asigura comportamentul concurent al serverului. Vom activa si keepalive[3] pentru conexiunile TCP dintre clienti si server pentru ca serverul sa poata indentifica toti clientii picati.

Pentru datele organizatorice ale concursului, serverul pune la dispozitie fisierele de configurare in format JSON[1]: server_config.json, contenders.json si problems.json ce vor contine date cu privire la modul de desfasurare a concursului respectiv lista concurentilor ce ar putea lua parte la acest concurs si lista de probleme ce pot fi selectate pentru acest concurs.

3.2 Diagrama aplicatiei

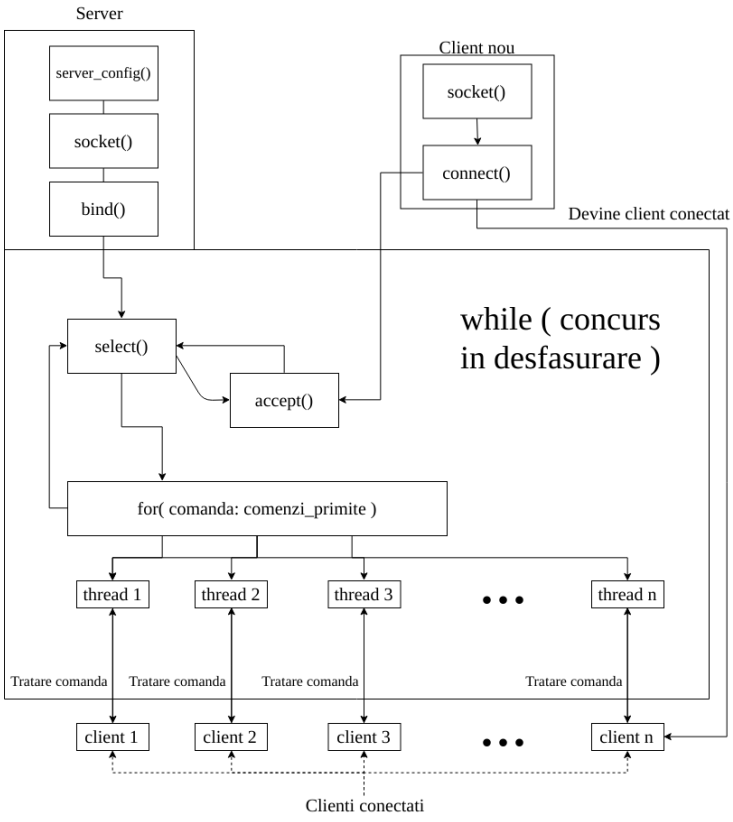

Fig. 1. Diagrama ce prezinta modul de functionare a serverului si modul in care acesta trateaza cererile clientilor sai.

Functia server_config() se ocupa de initializarea datelor programului cu ajutorul fisierelor de configurare ale serverului specificate mai sus, in special a fisierului server_config.json.

Atat inchiderea socket-urilor clientilor cat si compilarea si notarea solutiilor concurentului au loc in cadrul tratarii comenzilor clientilor de catre threaduri. Inchiderea socket-ului unui client are loc atunci cand serverul primeste ca mesaj comanda LOGOUT de la client, moment in care aplicatia client se deconecteaza si ea de la server urmand sa fie inchisa. Notarea solutiilor are loc odata cu trimiterea comenzii SUBMIT de catre client, fie din proprie initiativa (concurentul a terminat de rezolvat problemele si doreste sa le trimita), fie fortat de catre server (serverul cere trimiterea acestei comenzi de catre toti clientii sai odata cu terminarea timpului pus la dispozitie pentru rezolvarea problemelor). Restul comenzilor sunt simple schimburi de informatii intre client si server.

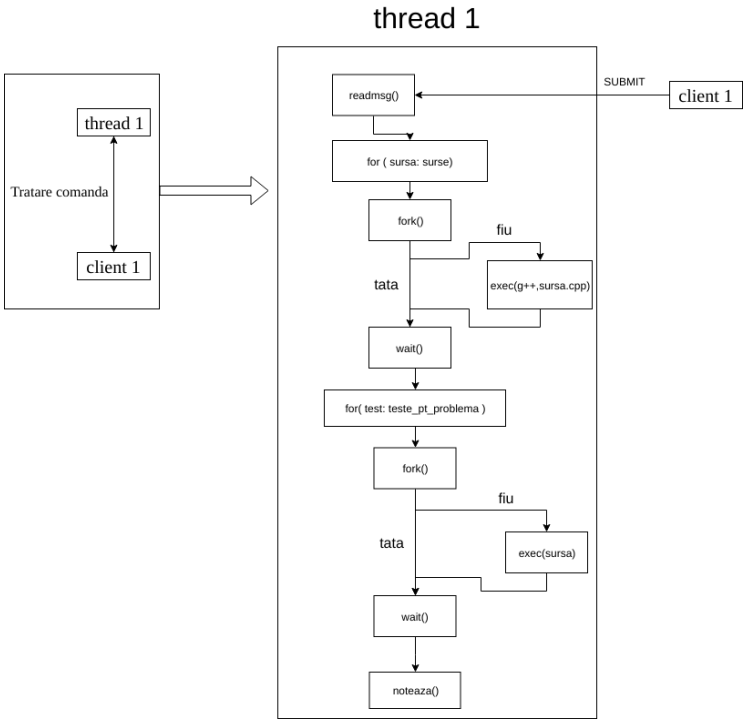

Fig. 2. Diagrama ce prezinta modul in care are loc evaluarea automata a surselor trimise de catre un client.

4 Detalii de implementare

Modul in care are loc schimbul de mesaje este descris in "OIOprot.h" si "OIOprot.cpp". Aceste fisiere descriu un protocol de functii de trimitere si primire a

comenzilor dintre server si client si tipurile de mesaje ce pot aparea in cadrul acestui proiect. Nu contin insa si codul necesar tratarii acestor comenzi ci doar schimbul de informatii si cum trebuie facut acesta.

Declararea tipurilor de mesaje din proiect:

```
enum COMMAND: int{LOGIN,REJECT,ACCEPT,
    CONTEST_INFO,CONTEST_INFO_REQ,
    TIME,TIME_REQ,CONTEST_ANS,CONTEST_ANS_REQ,
    CONTEST_REZ,CONTEST_REZ_REQ,LOGOUT};
```

Toate mesajele sunt de forma: dimensiune mesaj, continut mesaj. Trimitem mai intai sizeof(size_t) bytes ce reprezinta dimensiunea mesajului (fie acesta = length) si apoi length bytes ce reprezinta continutul mesajului. Pentru acest lucru se folosesc functiile "readmsg" si "writemsg" descrise mai jos:

```
void writemsg(int to,const void* what,size_t length)
{
    if(what != nullptr)
    {
        char buff_len[sizeof(size_t)];
        bcopy(&length,buff_len, sizeof(size_t));

        ssize_t n;
        size_t sum = 0;
        while(sizeof(size_t)-sum>0) { // trimitem dimensiunea
    mesajului
            n = write(to,buff_len+sum,sizeof(size_t)-sum);
            if (n == -1)
                FAT_ERROR(write);
            sum+=n;
        }

        sum = 0;
        while(length-sum>0) { // trimitem mesajul
            n = write(to,(const char*)what+sum,length-sum);
            if (n == -1)
                FAT_ERROR(write);
            sum+=n;
        }
    }
}

size_t readmsg(int from,void*& in)
{

    size_t length;
    char buff_len[sizeof(size_t)];
    bcopy(&length,buff_len, sizeof(size_t));

    ssize_t n;
    size_t sum = 0;
```

```
36      while(sizeof(size_t)-sum>0) { // citim cat de lung va fi
    mesajul
37          n = read(from,buff_len+sum,sizeof(size_t)-sum);
38          if (n == -1)
39              FAT_ERROR(write);
40          sum+=n;
41      }
42
43      length = *((size_t*)buff_len);
44
45      if(in == nullptr)in = malloc(length);
46        // alocam spatiu suficient pentru stocarea mesajului
47        // acest lucru are loc doar cand in == nullptr, iar
48        // spatiul alocat astfel va fi eliberat prin apelul
49        // functiei free() de catre cine apeleaza readmsg
50
51      sum = 0;
52      while(length-sum>0) { // citim mesajul
53          n = read(from,(char*)in+sum,length-sum);
54          if (n == -1) {
55              free(in);
56              FAT_ERROR(write);
57          }
58          sum+=n;
59      }
60      return length;
61  }
```

Singurele fisiere trimise intre client si server sunt fisiere ce descriu fie enunturi pentru probleme de olimpiada (server ⇒ client) fie rezolvari C++ pentru problemele primite (client ⇒ server). Astfel, dimensiunea acestor fisiere nu va fi foarte mare iar timpul de transmitere al acestora va fi relativ mic. Totodata nu avem comenzi urgente intre server si client care sa necesite sa fie trimise chiar si in timpul trimiterii unui fisier. Din aceste motive vom folosi acelasi socket atat pentru trimiterea fisierelor cat si a comenzilor, trimiterea unui sau mai multor fisiere fiind precedata de o comanda specifica. Functiile pentru trimiterea si receptionarea continutului unui fisier sunt:

```
1   void sendfile(int to,const char *file) {
2       FILE * g;
3       // deschidem pentru citire
4       if((g = fopen(file,"r")) == nullptr)
5           FAT_ERROR(fopen);
6       // trimitem cate FILE_BUFF bytes
7       char buff[FILE_BUFF];
8       while(fgets(buff,FILE_BUFF,g) != nullptr){
9           writemsg(to,buff,(strlen(buff) + 1)* sizeof(char));
10      }
11      // trimitem un mesaj individual de dimensiune 1
```

```
12      // ce va contine '\0' pentru a anunta sfarsitul
        transmiterii
13      char eof = '\0';
14      writemsg(to,&eof, sizeof(char));
15
16      fclose(g);
17  }
18
19  void receivefile(int from,const char *file) {
20      FILE *g;
21      size_t length = strlen(file);
22      char* temp = (char*)malloc(length + 1);
23      // daca este nevoie de crearea a noi directoare
24      // pentru a salva fisierul, le cream
25      bcopy(file,temp,length+1);
26      create_path(dirname(temp));
27      free(temp);
28      // deschidem pentru scriere
29      if((g = fopen(file,"w")) == nullptr)
30          FAT_ERROR(fopen);
31      // citim continutul fisierului mesaj cu mesaj
32      char *buff = (char*)malloc(FILE_BUFF);
33      do{
34          void *pVoid = buff;
35          length = readmsg(from,pVoid);
36          buff = (char*)pVoid;
37          if(!(length == sizeof(char) && *buff == '\0'))
38              fprintf(g,"%s",buff);
39      }while(!(length == sizeof(char) && *buff == '\0')); //
        pana intalnim mesajul de oprire
40
41      free(buff);
42      fclose(g);
43  }
```

Exemplu de functii pentru transmiterea si primirea unei comenzi din cele enumerate mai sus:

```
1   void sendLOGIN(int to, char *name, char *pass, int code) {
2       COMMAND login = LOGIN;  // anuntam tipul de comanda
3       writemsg(to,&login,sizeof(COMMAND));
4       // trimitem datele in mesaje individuale
5       writemsg(to,name,(strlen(name) + 1)* sizeof(char));
6       writemsg(to,pass,(strlen(pass) + 1)* sizeof(char));
7       writemsg(to,&code,sizeof(int));
8   }
9
10  void receiveLOGIN(int from, char *name, char *pass, int &code)
        {
11      // tipul mesajului a fost deja citit, mai trebuie citite
        datele
```

```
12      // citim numele
13      void *pVoid = name;
14      readmsg(from,pVoid);
15      name = (char*)pVoid;
16      // citim parola
17      pVoid = pass;
18      readmsg(from,pVoid);
19      pass = (char*)pVoid;
20      // citim codul concursului
21      pVoid = &code;
22      readmsg(from,pVoid);
23  }
```

Pentru prelucrarea datelor din fisierele de configurare de tip JSON am folosit libraria "nlohmann/json" sau "JSON for Modern C++"[2]. Aceasta prelucrare are loc la inceputul programului o singura data pentru a initializa parametrii serverului pentru concursul pe care il va rula.

5 Concluzii

Solutia prezentata permite derularea unuei olimpiade online a carei mod de organizare este descris in prealabil cu ajutorul fisierelor de configurare. Posibile imbunatatiri ale acestei solutii ar fi:

- adaugarea de comenzi in cadrul aplicatiei server pentru configurarea concursului pentru ca administratorul serverului (organizatorul concursului) sa poata mai usor modifica aceste date la fiecare noua rulare a apicatiei;
- trimiterea regulata a mesaje de la server la client in legatura cu timpul ramas pana la incheierea concursului (momentan aceste informatii se trimit doar la cererea clientului);
- adaugarea posibilitatii de a urmari cand si de la ce IP se conecteaza / deconecteaza un concurent pe parcursul concursului pentru a indentifica posibilele incercari de frauda;

References

1. Informatii despre formatul JSON https://www.json.org/
2. Libraria si documentatia librariei nlohmann/json - "JSON for Modern C++" aflata sub licenta MIT https://github.com/nlohmann/json
3. Informatii despre optiunea KEEPALIVE http://tldp.org/HOWTO/TCP-Keepalive-HOWTO/overview.html