OnlineInfoOlympiad

Ghionea Petru-Daniel

Universitatea "Alexandru Ioan Cuza" din Iasi

Abstract. OnlineInfoOlympiad este o aplicație ce simulează o olimpiadă online de informatică, permitând conectarea mai multor utilizatori folosind un model client-server. Aplicația analizează soluțiile primite de la utilizatori și transmite înapoi nota obținută.

Keywords: Olimpiada online, Model client-server, Corectare automată

1 Introducere

Scopul acestui proiect este crearea unui model client-server interactiv și ușor de utilizat pentru o olimpiadă online de informatică. Obiectivele acestui proiect sunt:

- Automatizarea: Dezvoltarea unui sistem de evaluare automată a soluțiilor trimise de utilizatori.
- Feedback-ul: Oferirea unui feedback utilizatorilor cu privire la corectitudinea soluțiilor acestora și punctajului obținut.
- Extensibilitatea: Crearea unei arhitecturi ce permite adăugarea de noi funcționalități și îmbunătățiri ulterioare.

2 Tehnologii aplicate

Această aplicație folosește modelul TCP/IP (Transmission Control Protocol/Internet Protocol) utilizat pentru a permite comunicarea între dispozitivele dintr-o rețea. Format din patru nivele și anume:

1. Nivelul fizic: Asigură conectarea host-ului la rețea.
2. Nivelul rețea: Permite gazdelor să emită pachete în orice rețea; pachete care vor circula independent până la destinație.
3. Nivelul transport: Asigură realizarea comunicării între gazda sursă și gazda destinație.
4. Nivelul aplicație: Conține protocoale de nivel înalt și este cel mai apropiat de utilizator, oferind o interfață pentru aplicația care utilizează rețeaua.

La nivelul transport, dintre protocolul TCP și protocolul UDP, chiar dacă protocolul UDP este mai rapid, acesta nu oferă garanții privind livrarea sau ordinea datelor, nu are control de flux, deci nu așteaptă o confirmare de primire de la destinatar înainte de a transmite următorul pachet și nu este un protocol orientat-conexiune, deci nu stabilește o conexiune înainte de a trimite date.

Toate aceste aspecte sunt necesare pentru a simula un concurs de tip olimpiadă, deci a fost ales protocolul TCP.

3 Structura aplicației

Pentru modelarea acestei aplicații au fost folosite următoarele concepte:

- Sockets: Sunt utilizate funcțiile din `<sys/socket.h>` pentru a crea și gestiona socket-uri ce permit comunicarea între server și client.
- Forking: Este utilizată funcția `fork()` în server pentru a crea procese fiu ce vor comunica cu un client în parte, permitând astfel acceptarea simultană a mai multor clienți.
- Manipularea de fișiere: Serverul citește conținutul fișierului de configurare și al problemei alese folosind funcții precum `fopen` sau `fread`.
- Manipularea de directoare: Este utilizată o funcție custom pentru a determina calea completă a unui fișier până la un anumit director.
- Generare aleatoare: Serverul alege în mod aleatoriu problema ce urmează a fi rezolvată de către client, utilizând funcția `rand()` pentru a genera un număr aleatoriu.
- Aplicarea conceptelor de concurență: serverul folosește primitiva `select()` pentru a gestiona conectarea clienților la server într-un anumit timp.
- Gestionarea timpului: sunt utilizate primitivele `time()` și `clock()` pentru a monitoriza timpul de rezolvare al clientului.
- Memorie partajată: serverul folosește un flag partajat în memorie pentru a trimite problema ce trebuie rezolvată clienților în același timp.
- Compilare și execuție: serverul compilează și rulează soluția clientului folosind apeluri de sistem.
- Validarea rezultatelor: serverul compară rezultatul obținut de la client cu soluția corectă și acordă o notă în funcție de timpul de executie, corectitudinea soluției și timpul de upload.

De asemenea, a fost creată o diagramă pentru a conceptualiza mai ușor aplicația(vezi figura 1).

4 Aspecte de implementare

Aplicația va urma următorul protocol de comunicare:

- Conectarea: Serverul așteaptă conexiuni la portul 2000 folosind primitiva `listen`; iar clientul se conectează la adresa serverului și portul specificate în linia de comandă folosind primitiva `connect`.De asemenea, este utilizată primitiva `'select'` pentru a aloca un anumit timp de conectare. Dacă acest timp expiră, problema ce trebuie rezolvată va fi trimisă clienților și nu vor mai fi acceptate conexiuni.
- Inițierea comunicării: După conectare, serverul trimite problema selectată clientului împreună cu timpul alocat rezolvării și va genera fișierul în care se va scrie rezolvarea.

- Comunicarea propriu-zisă: Clientul va transmite serverului comenzi și va citi răspunsurile primite folosind primitivele `read` și `write`. Serverul va utiliza aceleași primitive pentru a citi comenzile și a transmite răspunsurile. De asemenea, clientul va avea lista de comenzi disponibile și va aștepta introducerea unei comenzi de către utilizator.
- Gestionarea timpului alocat rezolvării: După expirarea timpului alocat, serverul va schimba permisiunile de acces la fișierul în care se află rezolvarea, va copia rezolvarea clientului, o va compila și corecta. De asemenea, corectarea va începe și dacă utilizatorul introduce comanda 'submit'.Corectarea constă în compararea fișierului de output al clientului cu fișierul de output corect. Se va asigna o notă în funcție dacă fișierul de output este corect, parțial corect sau incorect; precum și în funcție de timpul de execuție și timpul de upload.
- Încheierea conexiunii: Dacă utilizatorul introduce comanda 'exit', serverul va opri conexiunea cu acel client.

Într-un scenariu real, aplicația ar putea fi utilizată în următorul mod:

Un profesor coordonator ce va rula aplicația server va modifica fișierul de configurare stabilind numărul maxim de clienți, timpul alocat pentru rezolvarea problemei alese și timpul alocat conectării.. Elevii ce participă la concurs vor rula pe mașinile proprii aplicația client specificând în linia de comandă adresa IP și portul, atât timp cât nu depășesc timpul alocat conectării.. Aplicația server va alege în mod aleator o problemă și va trimite clientilor informații despre aceasta, precum și timpul alocat rezolvării.

Elevii vor rezolva problema în timpul alocat și vor scrie rezolvarea în fișierul denumit 'rezolvare.cpp' creat de server. După terminarea rezolvării, elevii vor introduce comanda 'submit' pentru a notifica serverul să înceapă corectarea. După introducerea comenzii 'submit', elevul respectiv nu va mai avea acces la fișierul 'rezolvare.cpp' pentru a face modificări. De asemenea, după expirarea timpului alocat, comanda 'submit' va fi apelată în mod automat. Serverul va corecta soluțiile și va nota fiecare elev în parte. După primirea rezultatului obținut, elevul introduce comanda 'exit' pentru a se deconecta.

Secțiuni de cod importante:

Listing 1.1. Functia pentru obtinerea caii complete a unui fisier pana la un director specificat

```
1  char* getFullPathUntilDir(const char* filePath, const
       ↪ char* targetDir)
2  {
3    char currentDir[MAX_PATH_LENGTH];
4    char fileName[MAX_PATH_LENGTH];
5    char fileDir[MAX_PATH_LENGTH];
6
7    // Obtine directorul de lucru curent
8    if (getcwd(currentDir, MAX_PATH_LENGTH) == NULL)
9    {
10     perror("[server]Eroare la gasirea directorului
         ↪ curent de lucru.\n");
```

```
11      return NULL;
12    }
13
14    // Creeaza calea completa a fisierului
15    snprintf(fileName, MAX_PATH_LENGTH, "%s/%s",
         ↪ currentDir, filePath);
16
17    // Copiaza calea completa intr-o alta variabila
18    strncpy(fileDir, fileName, MAX_PATH_LENGTH);
19
20    // Extrage directorul parinte al fisierului
21    char* parentDir = dirname(fileDir);
22
23    // Cauta directorul target in interiorul directorului
         ↪  parinte
24    char* targetDirPos = strstr(parentDir, targetDir);
25
26    // Verifica daca directorul target a fost gasit
27    if (targetDirPos != NULL)
28    {
29      // Calculeaza lungimea subsirului de la inceputul
           ↪ directorului parinte pana la sfarsitul
           ↪ directorului target
30      size_t length = targetDirPos - parentDir + strlen(
           ↪ targetDir);
31
32      // Aloca memorie pentru subsir
33      char* fullPath = malloc(length + 1);
34
35      // Verifica daca alocarea de memorie a reusit
36      if (fullPath == NULL)
37      {
38        perror("[server]Eroare la malloc.\n");
39        return NULL;
40      }
41
42      // Copiaza subsirul in memoria nou alocata
43      strncpy(fullPath, fileName, length);
44
45      // Adauga terminatorul de sir '\0'
46      fullPath[length] = '\0';
47
48      // Returneaza sirul alocat dinamic continand calea
           ↪ pana la directorul target
49      return fullPath;
```

```
50    }
51    else
52    {
53      // Afiseaza un mesaj de eroare daca directorul
          ↪ target nu a fost gasit
54      printf("[server]Eroare: Directorul target %s nu a
          ↪ fost gasit.\n", targetDir);
55      return NULL;
56    }
57  }
```

Listing 1.2. Functia 'submit'

```
1  int submit(int submit_time)
2  {
3    ssize_t bytes_read;
4    char buffer[MAX_CONTENT], solution[100];
5    const char *filePath = "Proiect/Server/server.c";
6    const char *targetDir = "Proiect";
7    const clientDir[100];
8    int nota;
9
10   // Construieste calea catre fisierul solutiei
       ↪ clientului
11   sprintf(clientDir, "/Client%d", contorclienti);
12   char *fullPath = getFullPathUntilDir(filePath,
       ↪ targetDir);
13   strcat(fullPath, clientDir);
14   strcat(fullPath, "/rezolvare.cpp");
15
16   // Creeaza un fisier pentru solutia clientului
17   sprintf(solution, "solutie_client%d.cpp",
       ↪ contorclienti);
18   int fd_destination = open(solution, O_CREAT | O_RDWR,
       ↪  0777);
19   int fd_source = open(fullPath, O_RDONLY);
20
21   // Copiaza continutul solutiei clientului intr-un
       ↪ fisier nou
22   while ((bytes_read = read(fd_source, buffer,
       ↪ MAX_CONTENT)) > 0)
23   {
24     write(fd_destination, buffer, bytes_read);
25   }
26
27   // Inchide descriptorii de fisiere
28   close(fd_source);
```

```
29   close(fd_destination);
30
31   // Efectueaza evaluarea solutiei trimise
32   nota = corectare(submit_time);
33
34   // Returneaza rezultatul evaluarii
35   return nota;
36 }
```

Listing 1.3. Functia ‘corectare‘

```
1  int corectare(int submit_time)
2  {
3    FILE *fd_out_corect, *fd_out_client;
4    clock_t start_execution, end_execution;
5    double execution_time, one_fourth, two_fourths,
       ↪ three_fourths, four_fourths;
6    char solution[100], compiled[100],
       ↪ compilation_command[100], execution_command
       ↪ [100], nume_out[100];
7    char buffer1[100], buffer2[100];
8    int nota, ok = 1, partial_ok = 0;
9
10   // Construieste numele fisierelor si comenzilor
       ↪ pentru compilare si executie
11   sprintf(solution, "solutie_client%d.cpp",
       ↪ contorclienti);
12   sprintf(compiled, "solutie_client%d", contorclienti);
13   sprintf(compilation_command, "g++ -Wall %s -o %s",
       ↪ solution, compiled);
14   sprintf(execution_command, "./%s", compiled);
15
16   // Calculeaza intervalele de timp pentru evaluare
17   one_fourth = secunde_rezolvare / 4;
18   two_fourths = 2 * secunde_rezolvare / 4;
19   three_fourths = 3 * secunde_rezolvare / 4;
20   four_fourths = 4 * secunde_rezolvare;
21
22   // Compileaza solutia clientului
23   int compilation_result = system(compilation_command);
24
25   if (compilation_result == 0)
26   {
27     start_execution = clock();
28
29     // Executa solutia clientului
30     int execution_result = system(execution_command);
```

```
31
32      end_execution = clock();
33      execution_time = (((double)(end_execution -
           ↪ start_execution)) / CLOCKS_PER_SEC) *
           ↪ 10000.0;
34
35      if (execution_result == 0)
36      {
37        // Deschide fisierul de output corect si fisierul
             ↪  de output de la client
38        fd_out_corect = fopen("correct_output.out", "r");
39        switch (randomP)
40        {
41        case 1:
42          sprintf(nume_out, "recyclebin%d.out",
               ↪ contorclienti);
43          fd_out_client = fopen(nume_out, "r");
44          break;
45        case 2:
46          sprintf(nume_out, "rufe%d.out", contorclienti);
47          fd_out_client = fopen(nume_out, "r");
48          break;
49        case 3:
50          sprintf(nume_out, "tairos%d.out", contorclienti
               ↪ );
51          fd_out_client = fopen(nume_out, "r");
52          break;
53        }
54
55        while (fgets(buffer1, sizeof(buffer1),
             ↪ fd_out_corect) != NULL && fgets(buffer2,
             ↪ sizeof(buffer2), fd_out_client) != NULL)
56        {
57          // Compara liniile din fisierele de output
58          if (strcmp(buffer1, buffer2) == 0)
59          {
60            partial_ok = 1;
61          }
62          else if (strcmp(buffer1, buffer2) != 0)
63          {
64            ok = 0;
65            break;
66          }
67        }
68
```

```
69        // Asigneaza nota in functie de rezultatul
             ↪ compararii
70        if (ok == 1)
71        {
72          if (execution_time <= 0.5 && submit_time <=
               ↪ one_fourth)
73            nota = 10;
74          else if (execution_time <= 1.0 && (submit_time
               ↪ <= one_fourth || submit_time <=
               ↪ two_fourths))
75            nota = 9;
76          else if (execution_time <= 2.0 && (submit_time
               ↪ <= one_fourth || submit_time <=
               ↪ two_fourths || submit_time <=
               ↪ three_fourths))
77            nota = 8;
78          else if (execution_time <= 3.0 && (submit_time
               ↪ <= one_fourth || submit_time <=
               ↪ two_fourths || submit_time <=
               ↪ three_fourths || submit_time <=
               ↪ four_fourths))
79            nota = 7;
80          else
81            nota = 6;
82        }
83        else if (partial_ok == 1)
84        {
85          nota = 5;
86        }
87        else
88        {
89          nota = 4;
90        }
91      }
92      else
93      {
94        // Afiseaza mesaj de eroare daca executia a esuat
95        char run_error[100];
96        sprintf(run_error, "[server]Eroare la executia
             ↪ clientului %d\n", contorclienti);
97        perror(run_error);
98      }
99    }
100   else
101   {
```

```
102       // Solutia nu a putut fi compilata
103       nota = 3;
104     }
105
106     // Inchide dexcriptorii de fisiere si sterge
          ↪ fisierele create temporar
107     fclose(fd_out_corect);
108     fclose(fd_out_client);
109     remove(solution);
110     remove(compiled);
111     remove(nume_out);
112
113     // Returneaza nota
114     return nota;
115 }
```

5 Concluzii

Soluția propusă ar putea fi îmbunătățită astfel:

– Adăugarea unui clasament al elevilor în funcție de nota obținută.

6 Referințe bibliografice

1. Site-ul cursului de rețele de calculatoare: https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
2. Site-ul profesorului de laborator: https://www.andreis.ro/teaching/computernetworks
3. Site-ul folosit pentru a crea diagrama: https://sequencediagram.org/

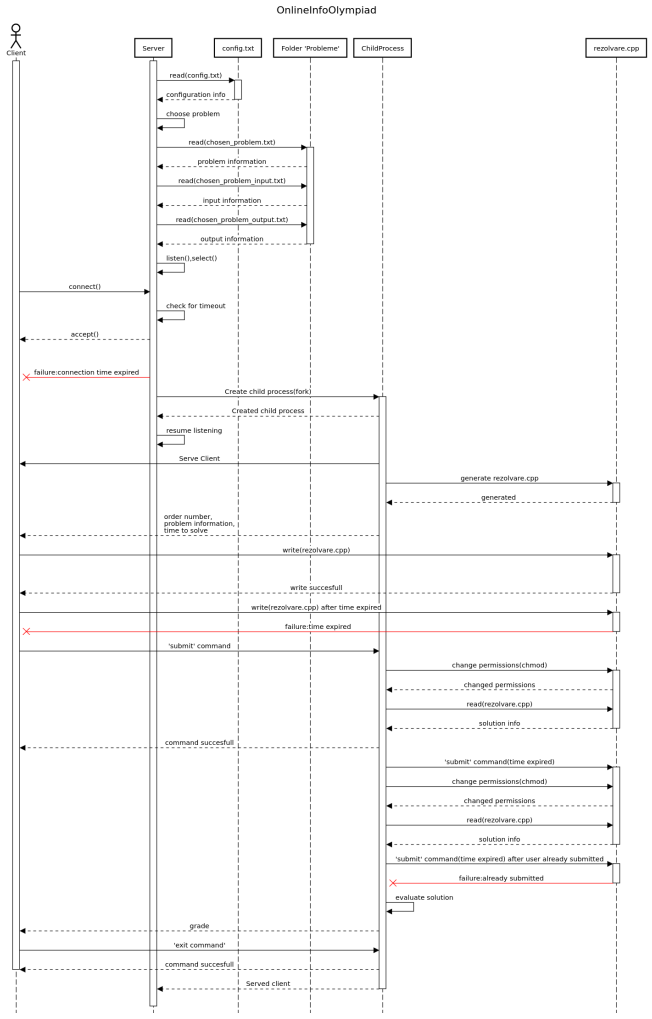

Fig. 1. Diagrama aplicatiei