Competitie de Informatica Online

Virlescu Ioan

Universitatea Alexandru-Ioan Cuza, Facultatea de Informatica

Abstract. Aceasta lucrare prezinta implementarea unui model client-server pentru organizarea unei competitii de informatica online. Modelul utilizat integreaza socket-uri TCP pentru o comunicare intre server si clienti, impreuna cu multithreading pentru concurenta. Serverul distribuie cerinte de programare, colecteaza solutii, si evalueaza rezultatele trimise de clienti. Clientul poate comunica cu serverul folosind comenzi predefinite pentru a interactiona eficient. Obiectivul principal este de a demonstra o baza solida pentru dezvoltarea ulterioara a sistemelor distribuite utilizate in competitii automatizate.

1 Introducere

Competitiile de informatica online reprezinta un mediu excelent pentru dezvoltarea competentelor in programare si gandire algoritmica. Pentru a facilita organizarea eficienta a acestor competitii, este necesar un sistem automatizat care sa permita interactiunea intre participanti si organizatori.

Aceasta lucrare prezinta un model client-server care permite:

- Distribuirea cerintelor problemelor de programare catre participanti.
- Colectarea, compilarea si evaluarea automata a solutiilor trimise de clienti.
- Gestionarea conexiunilor multiple in mod concurent.
- Crearea si afisarea clasamentelor actuale si generale.

Serverul este responsabil de procesarea comenzilor, gestionarea resurselor necesare si evaluarea automata a solutiilor, in timp ce clientii interactioneaza cu acesta printr-un protocol simplu si clar. Scopul principal este de a demonstra eficienta unui astfel de sistem intr-un mediu distribuit.

2 Tehnologii Aplicate

2.1 Protocolul TCP

TCP (Transmission Control Protocol) a fost ales pentru comunicarea intre server si clienti datorita urmatoarelor avantaje:

- **Fiabilitate:** TCP garanteaza livrarea pachetelor in ordinea corecta, ceea ce este esential pentru transferul comenzilor si raspunsurilor.
- **Conexiuni persistente:** Comunicarea persista pe durata sesiunii, eliminand nevoia de reconectare pentru fiecare comanda.
- **Controlul erorilor:** TCP gestioneaza retransmiterea pachetelor pierdute, asigurand integritatea datelor transferate.

2.2 Multithreading

Serverul utilizeaza multithreading pentru a gestiona mai multi clienti simultan. Fiecare client este deservit de un fir de executie dedicat, astfel incat activitatile unui client nu afecteaza altii.

2.3 SQLite pentru Baza de Date

Pentru stocarea informatiilor despre utilizatori, probleme, teste, solutii si clasamente, proiectul foloseste o baza de date SQLite. Aceasta ofera:

- Persistenta datelor, permitand salvarea istoricului competitiilor.
- Operatii rapide de interogare si actualizare.
- Flexibilitate in extinderea schemelor bazei de date.

2.4 Socket-uri BSD

Socket-urile BSD sunt folosite pentru comunicarea bidirectionala intre client si server. Acestea ofera o interfata standardizata pentru transferul de date folosind protocolul TCP.

3 Structura Aplicatiei

3.1 Modelul Aplicatiei

Aplicatia este formata din doua componente principale:

1. **Serverul:**
   - Asteapta conexiuni de la clienti.
   - Creeaza un thread dedicat pentru fiecare client conectat.
   - Proceseaza comenzile primite si trimite raspunsurile corespunzatoare.
   - Evalueaza solutiile clientilor prin compilare si rularea testelor asociate.
2. **Clientul:**
   - Trimite comenzi catre server.
   - Primeste raspunsuri si afiseaza informatiile corespunzatoare utilizatorului.
   - Permite trimiterea de solutii catre server.

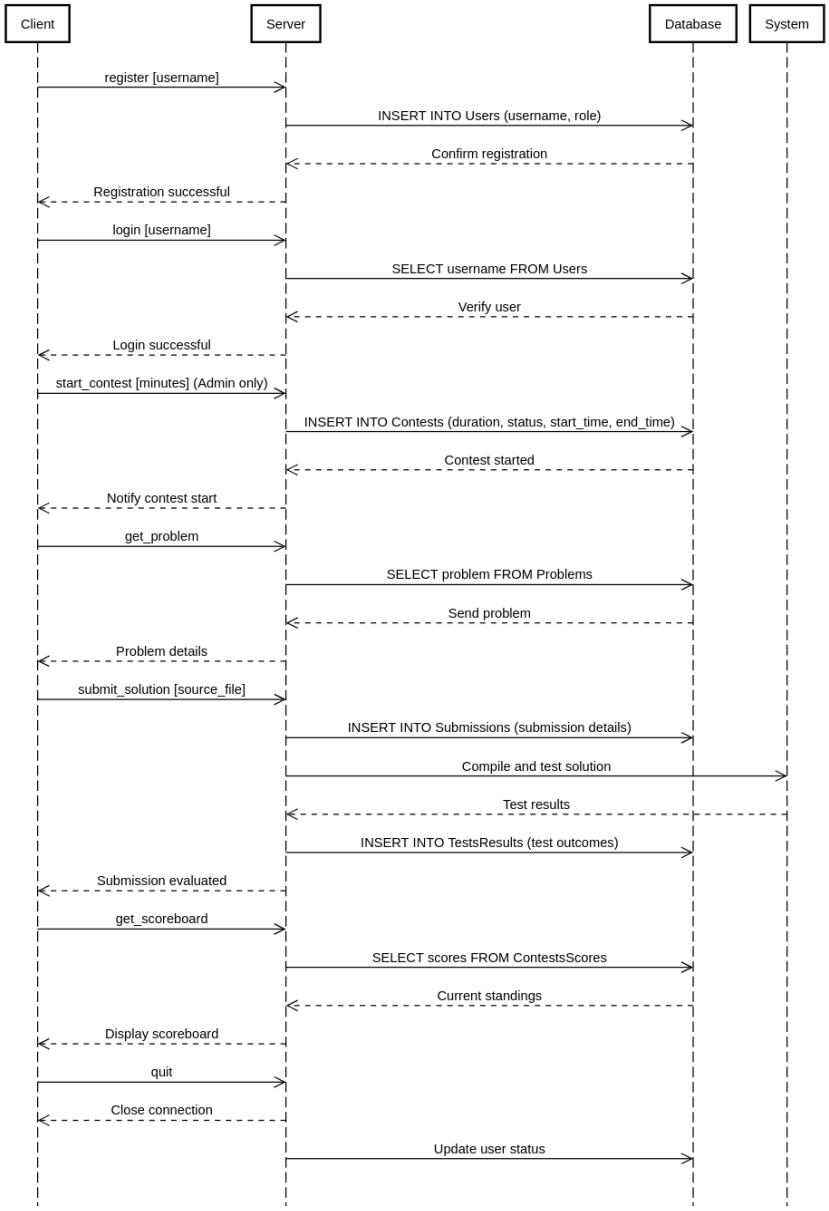

Fig. 1. Diagrama pentru modelul client-server.

4 Aspecte de Implementare

Sectiuni de cod

4.1 Gestionarea Concurentei pe Server

Serverul foloseste thread-uri pentru a gestiona clientii in mod concurent. Codul pentru deservirea unui client este:

```
void *raspunde(void *arg)
{
    int clientSocket = *(int *)arg;
    free(arg);

    while (1)
    {
        char mesaj[1024] = {0};
        int bytesReceived = recv(clientSocket, mesaj, 1024, 0);
        if (bytesReceived <= 0)
        {
            printf("Clientul s-a deconectat.\n");
            break;
        }

        if (strncmp(mesaj, "start_contest", 13) == 0)
        {
            // Pornirea unui concurs
        }
        else if (strncmp(mesaj, "submit_solution", 15) == 0)
        {
            // Evaluarea solutiei clientului
        }
        else if (strcmp(mesaj, "quit") == 0)
        {
            printf("Clientul a inchis conexiunea.\n");
            break;
        }
    }

    close(clientSocket);
    return NULL;
}
```

Clientul: Transmiterea cererilor

```
// Exemplu de cod pentru client
while (1)
    {
        FD_ZERO(&read_fds);
        FD_SET(STDIN_FILENO, &read_fds);
        FD_SET(sd, &read_fds);
```

```
        int max_fd = (STDIN_FILENO > sd) ? STDIN_FILENO : sd;

        if (select(max_fd + 1, &read_fds, NULL, NULL, NULL) < 0)
        {
            perror("Eroare la select.\n");
            break;
        }

        if (FD_ISSET(sd, &read_fds))
        {
            memset(mesaj, 0, sizeof(mesaj));
            if (read(sd, mesaj, sizeof(mesaj)) <= 0)
            {
                perror("Eroare la citirea de la server.\n");
                break;
            }
            else if (strncmp(mesaj, "exit", 4) == 0)
            {
                printf("[Raspunsul serverului]: Conexiunea a fost oprita. Progra
                break;
            }

            printf("[Raspunsul serverului]: %s\n", mesaj);
        }

        if (FD_ISSET(STDIN_FILENO, &read_fds))
        {
            fflush(stdout);
            memset(mesaj, 0, sizeof(mesaj));
            fgets(mesaj, sizeof(mesaj), stdin);
            mesaj[strcspn(mesaj, "\n")] = 0;

            if (write(sd, mesaj, strlen(mesaj)) <= 0)
            {
                perror("Eroare la scrierea catre sever.\n");
                break;
            }
        }
    }
```

4.2 Protocol la Nivel de Aplicație

- **REGISTER [username]:** Serverul înregistrează un utilizator nou în baza de date, dacă numele de utilizator nu există deja.
- **LOGIN [username]:** Serverul permite unui utilizator să se autentifice dacă există deja în baza de date și nu este conectat pe alt dispozitiv.

- **START_CONTEST [minutes]:** (doar pentru admin) Serverul pornește un concurs nou, cu durata specificată în minute, și notifică toți participanții conectați.
- **GET_PROBLEM:** Serverul trimite detaliile problemei curente de rezolvat (titlu, descriere, exemplu de input și output).
- **SUBMIT_SOLUTION [source_file]:** Serverul primește soluția utilizatorului, o compilează și o evaluează folosind testele asociate problemei.
- **GET_SCOREBOARD:** Serverul trimite clasamentul curent al participanților pentru concursul în desfășurare.
- **GET_TIME_LEFT:** Serverul trimite timpul rămas din concursul curent.
- **QUIT:** Serverul închide conexiunea cu clientul și actualizează statusul utilizatorului în baza de date.

4.3 Scenarii de Utilizare

1. **Înregistrarea unui utilizator:**
   - Clientul trimite comanda `REGISTER [username]`.
   - Serverul verifică dacă numele de utilizator există deja.
   - Dacă numele de utilizator este unic, serverul înregistrează utilizatorul în baza de date și confirmă înregistrarea.
2. **Autentificarea unui utilizator:**
   - Clientul trimite comanda `LOGIN [username]`.
   - Serverul verifică dacă utilizatorul există și nu este conectat pe alt dispozitiv.
   - Serverul confirmă autentificarea cu un mesaj de bun venit.
3. **Pornirea unui concurs:**
   - Adminul trimite comanda `START_CONTEST [minutes]`.
   - Serverul adaugă concursul în baza de date și trimite notificări tuturor participanților conectați.
   - Participanții primesc detalii despre concurs și problema curentă de rezolvat.
4. **Solicitarea unei probleme:**
   - Clientul trimite comanda `GET_PROBLEM`.
   - Serverul răspunde cu detaliile problemei curente (titlu, descriere, exemplu de input și output).
5. **Trimiterea unei soluții:**
   - Clientul trimite comanda `SUBMIT_SOLUTION [source_file]`.
   - Serverul primește fișierul sursă, îl compilează și îl evaluează folosind testele din baza de date.
   - Serverul trimite rezultatele evaluării către client și actualizează scorurile în baza de date.
6. **Vizualizarea clasamentului:**
   - Clientul trimite comanda `GET_SCOREBOARD`.
   - Serverul trimite clasamentul actualizat al participanților pentru concursul în desfășurare.
7. **Aflarea timpului rămas:**

   - Clientul trimite comanda `GET_TIME_LEFT`.
   - Serverul calculează și trimite timpul rămas din concursul curent.
8. **Deconectarea unui utilizator:**
   - Clientul trimite comanda `QUIT`.
   - Serverul închide conexiunea, actualizează statusul utilizatorului și confirmă închiderea conexiunii.

5 Concluzii

Acest proiect demonstreaza potentialul unui sistem client-server pentru organizarea competitiilor informatice. Prin utilizarea socket-urilor TCP, multithreading-ului si a unei baze de date SQLite, sistemul este capabil sa gestioneze eficient procesele critice ale unei competitii online. Dezvoltari ulterioare ar putea include:

- Optimizarea evaluarii solutiilor pentru a suporta un numar mai mare de clienti.
- Adaugarea unei interfete grafice pentru administrarea competitiei.
- Extinderea functionalitatilor bazei de date pentru analiza avansata a performantelor participantilor.

6 Referinte Bibliografice

References

1. Universitatea Alexandru Ioan Cuza: Protocoale, Socket-uri si Thread-uri. https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php
2. Springer: Guidelines for Authors of Proceedings Papers. https://resource-cms.springernature.com/springer-cms/rest/v1/content/3318/data/v6.
3. Pubs Opengroup https://pubs.opengroup.org/onlinepubs/7908799/xsh/select.html
4. SQLite https://www.sqlite.org/lang$_{d}ate func.html$