MERSUL TRENURILOR

Ghiarasim Alexandru(2A3)

Ianuarie 2024

1 Introducere

1.1 Viziune generală

Proiectul "Mersul Trenurilor"[Propunere Continental] vizează dezvoltarea unei aplicații server-client pentru gestionarea eficientă a informațiilor despre programul trenurilor. Scopul este de a oferi clienților date precise și actualizate privind sosirile, plecările și rutele disponibile între stații, precum și facilitarea comunicării despre posibile întârzieri ale anumitor trenuri.

1.2 Obiectivele proiectului

Printre obiectivele majore se numără: implementarea unui server eficient pentru gestionarea informațiilor despre programul trenurilor, implementarea unui sistem de comenzi care să satisfacă toate tipurile de cereri ale clienților, manipularea sigură a datelor din fișiere (XML) și actualizarea posibilelor întârzieri, precum și implementarea unei arhitecturi pentru a lucra cu oricât de mulți clienți.

2 Tehnologii aplicate

2.1 Protocolul TCP (Transmission Control Protocol)

Pentru implementarea comunicării server-client s-a optat pentru utilizarea protocolului TCP, alegere motivată de fiabilitatea pe care o asigură protocolul la livrare, datele fiind transmise fără erori și în ordinea primirii cererilor, și păstrarea datelor în siguranță (fișierele XML nu vor suferi modificări neașteptate).

2.2 Limbajul de programare

Am ales limbajul de programare C, bine-cunoscut pentru performanță și gestionarea eficientă a resurselor. Cu ajutorul bibliotecilor din C se pot concepe functionalități de bază, și anume: gestionarea socket-urilor, manipularea datelor, controlul erorilor, lucrul cu procese și fire de execuție.

2.3 Gestionarea fișierelor XML - librăria libxml2

Manipularea datelor se face folosind fișiere XML, ce vor servi drept "bază de date" a tuturor informațiilor despre trenuri. Acest mod de gestionare a informației facilitează extragerea datelor spre a fi trimise către clienți (de exemplu, programul unor anumite trenuri) și actualizarea întârzierilor. Toate funcționalitățile sunt implementate folosind biblioteca libxml2 în cadrul programului.

3 Structura aplicatiei

Arhitectura aplicației este concisă și bine organizată, fiecare componentă având rolul ei și asigurând transmiterea eficientă a datelor. fig.1 ilustrează structura protocolului TCP, evidențiind modul în care serverul servește clienții în mod concurent, utilizând thread-uri separate. Acest mod de operare asigură clienților răspunsuri imediate și evită așteptarea, fiecare solicitare primind un răspuns în concordanță cu cererea sa.

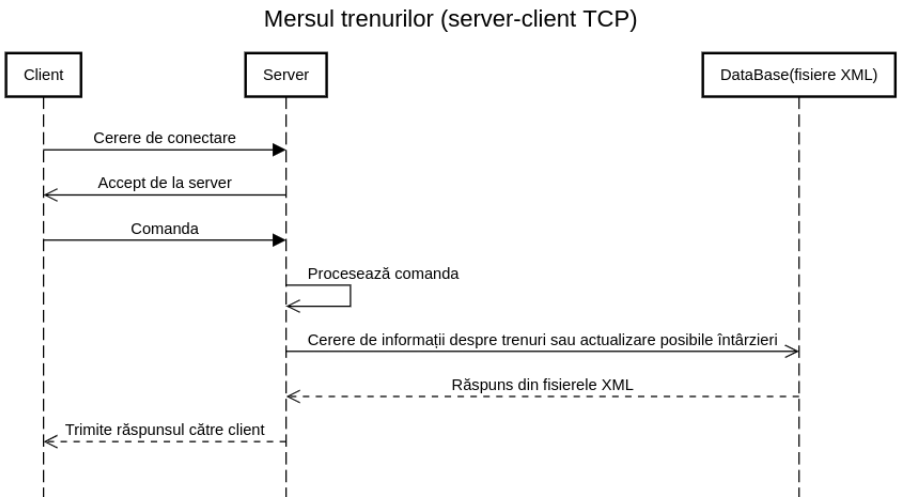

fig.1

După configurarea socket-ului pentru a aștepta conexiuni de la clienți, serverul inițiază o buclă de așteptare a acestora. Când un client primește accept(), prin apelul la primitiva "pthreadcreate()" se creează o conexiune separată cu serverul pentru a fi servit cu informații în mod concurent față de alți clienți și pentru a nu afecta celelalte conexiuni.

În fig.1, **clientul** solicită un string de la tastatură, prin comanda "read()", reprezentând o comandă care va fi ulterior validată și procesată de server. De exemplu, comanda "-help" este citită de la tastatură, trimisă prin primitiva write() la server, iar apoi primitiva read() blochează procesul până când poate citi răspunsul de la server. După efectuarea unui schimb de informații, procesul revine la început cu validarea unei alte comenzi de la tastatură.

Serverul are rolul de a primi string-ul de la client și primul pas este recunoașterea tipului comenzii, urmată de implementarea tipului de răspuns corespunzător. Dacă se primește o comandă necunoscută, aceasta este recunoscută imediat de către server și răspunsul corespunzător este trimis înapoi la client. Dacă șirul citit reprezintă o comandă existentă în protocolul implementat, și, în același timp sunt verificate argumentele, serverul inițiază o cerere către datele parsate din fisierul XML pentru obținerea informațiilor specifice tipului de comandă. De asemenea, serverul poate efectua și modificări corespunzătoare în componenta

fișierelor prin citirea adecvată a comenzii care specifică o anumită întârziere a unui tren.

```
int pid = fork();
if (pid == -1)
{
    perror("Error in fork.\n");
    return errno;
}
int server_opened = 1;
if (pid == 0)
{
    // fiu
    char message_primit[2000];
    while (1)…
    close(sd);
}
else
{
    //tata
    welcome(); // connected succesfully
    while (1)…
    kill(pid,SIGUSR1);
    close(sd);
}
```

fig.2

Figura 2 descrie structura aplicației client. Prin apelul fork(), se creează un proces suplimentar pentru a facilita primirea de mesaje în mod simultan cu citirea de la tastatură în client. Procesul părinte preia o comandă de la tastatură și o transmite către thread-ul din server responsabil cu procesarea clientului specific. În același timp, procesul copil citește constant din socket toate informațiile primite și le afișează pe ecran. Acest mod de organizare este foarte eficient în situații în care se folosește comanda "-delay:", deoarece atunci când un client raportează o altă oră de sosire sau plecare a unui tren, toți clienții trebuie să primească o notificare despre modificările din program.

Cuvintele cheie care fac trimitere la conceptele utilizate în modelarea aplicației includ: arhitectura server-client, protocol TCP, bază de date - fișiere XML, thread-uri și procese. Se poate menționa, de asemenea, că serverul oferă securitate datelor, fiecare operație fiind validată conform unor standarde specifice care nu afectează integritatea informațiilor referitoare la mersul trenurilor. De asemenea, modificările aduse programului trenurilor sunt realizate în timp real și pot fi observate de către ceilalți clienți.

4 Aspecte de implementare

În cadrul proiectului, implementarea este structurată în conformitate cu un protocol TCP la nivelul aplicației, asigurând o comunicare eficientă între server și clienți.

fig.3

Funcția reprezentată în fig.3 recunoaște tipul comenzii după șirul de caractere primit de la client și execută comanda specifică, fie transmitând date către client, fie actualizând baza de date (XML). Întregul protocol al comenzilor este dirijat de această funcție, având un rol esențial în codul sursă al aplicației.

```c
void raspunde(void *arg)
{
  char msg_from_client[100];
  struct thData tdL;
  tdL = *((struct thData *)arg);

  while (1)
  {
    // Citim mesajul de la client
    if (read(tdL.cl, msg_from_client, sizeof(msg_from_client)) <= 0)
    {
      printf("[Thread %d]\n", tdL.idThread);
      perror("Eroare la read() de la client.\n");
      break;
    }
    recognise_command(tdL, msg_from_client);
    if (strcmp(msg_from_client, "-quit") == 0)
      break;
    if (server_opened == 0)
      break;
  }
  remove_client(tdL.cl);
  close(tdL.cl);
}
```

fig.4

Funcția prezentată în fig.4 reprezintă o secțiune specifică de cod a serverului în care se gestionează comunicarea cu un client pe un thread separat. La

primirea fiecărei comenzi, aceasta este transmisă către funcția responsabilă cu recunoașterea tipului comenzii, după care se efectuează operațiunile corespunzătoare până la transmiterea răspunsului înapoi la client.

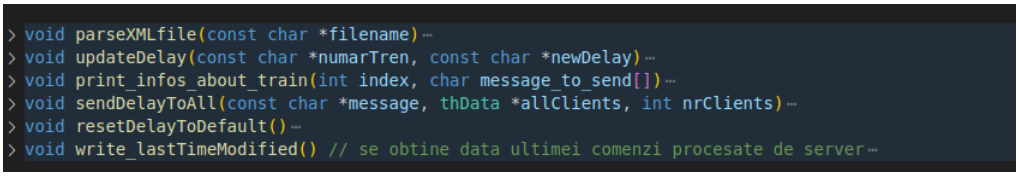

fig.5

În figura 5 se pot observa funcțiile declarate pentru a parsa datele din fișierul XML, (o dată la deschiderea serverului), pentru a modifica întârzierea unui tren în fișier, pentru a trimite mesaj la toți clienții despre o posibilă întârziere (funcția sendDelayToAll), precum și o funcție specială resetDelayToDefault() care se va asigura mereu că atunci când un tren ajunge la destinație, întârzierea este setată la 0, și astfel sunt verificate toate cazurile posibile.

Protocolul care va fi implementat va include procesarea comenzilor, interogarea bazei de date XML, gestionarea erorilor și asigurarea securității datelor transmise. Protocolul aplicației "Mersul Trenurilor" oferă utilizatorilor comenzi specifice pentru a furniza cât mai multe date filtrate în conformitate cu cererea efectuată. Acestea permit utilizatorilor să solicite informații despre comenzile disponibile, să obțină rute între stații, să vizualizeze mersul trenurilor, precum și să monitorizeze plecările și sosirile la o anumită stație, având posibilitatea de a încheia conexiunea cu serverul.

Linia din cod sugerează toate comenzile existente în protocol: char $allcommands[][100] =$ $" - help"," - ruta:"," - mersuri"," - plecari:"," - sosiri:"," - delay:"," - quit";$

Aplicația "Mersul Trenurilor" permite ca scenarii reale de utilizare consultarea în timp real a informații despre mersul trenurilor, planificarea de rute între stații, monitorizarea plecărilor și sosirilor la anumite stații și primirea de actualizări despre eventuale intârzieri sau modificări ale estimărilor de sosire. Utilizatorii pot încheia sesiunea de utilizare prin comanda "-quit" după ce au obținut informațiile dorite.

5 Concluzii

O primă îmbunătățire esențială pentru a face produsul livrabil constă în dezvoltarea unei interfețe grafice interactive, având ca obiectiv asigurarea unei experiențe mai plăcute utilizatorului. Extinderea funcționalităților, inclusiv adăugarea unor comenzi noi în protocol, cum ar fi planificarea călătoriilor și rezervarea de bilete, ar contribui semnificativ la crearea unei aplicații mai complexe și complete. De asemenea, integrarea unor sisteme externe, precum furnizarea de notificari în timp real privind întârzierile trenurilor, informații meteorologice sau conexiuni cu soferii, ar consolida imaginea și eficiența aplicației la utilizarea clientilor.

6 Referințe Bibliografice

- https://sequencediagram.org
- https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines
- https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
- https://en.wikipedia.org/wiki/TransmissionControlProtocol
- https://github.com/GNOME/libxml2
- https://en.wikipedia.org/wiki/Libxml2