Mersul trenurilor

Maftei Briana-Ştefania

January 9, 2024

1 Introducere

Timpul, cea mai importantă resursă, şi datele (fixate şi memorate), trebuie incorporate în informaţia care duce la îmbunătăţirea semnificativă a eficienţei feroviare. Această aplicaţie vine să susţină eficienţa feroviară printr-o interconexiune între prestator (server) si pasager (client). Clientul are posibilitatea de a afla detalii cu privire la statusul plecării şi sosirii (cu estimare) trenurilor, de a primi, de asemenea, menţiuni cu privire la posibile întârzieri şi de a trimite la rândul său în aplicaţie o astfel de informaţie.Aplicația evidențiază implementărea unui server concurent, car acceptă simultan clienți pentru procesarea cerințelor proiectului.

2 Tehnologiile utilizate

Pentru "Mersul trenurilor" aplicăm protocolul TCP în defavoarea celui UDP. În primul rând, acest protocol asigură livrarea datelor integral, în ordine. Serverul va trimite clientului sigur informaţia de care are nevoie, spre exemplu, despre trenul de care are nevoie şi nu despre altul. De aici vine în continuare şi al doilea avantaj: există confirmare că datele trimise au fost şi primite. Mai mult, se pot transmite date din ambele direcţii simultan şi serverul se poate ocupa de clienţi în paralel fiind un protocol orientat-conexiune. Protocolul UDP este neorientat-conexiune, iar cu acesta se poate transmite informaţia într-un timp mai bun, însă rapiditatea mai mare decât cea a protocolului TCP nu este suficientă când vine vorba de trimiterea cu erori/pierderi a informaţiei sau că nu se realizează primirea ei. Alegerea unui server TCP este mai avantajos aşadar, corectitudinea informaţiilor fiind esenţială în acest proiect pentru schimbul de date.

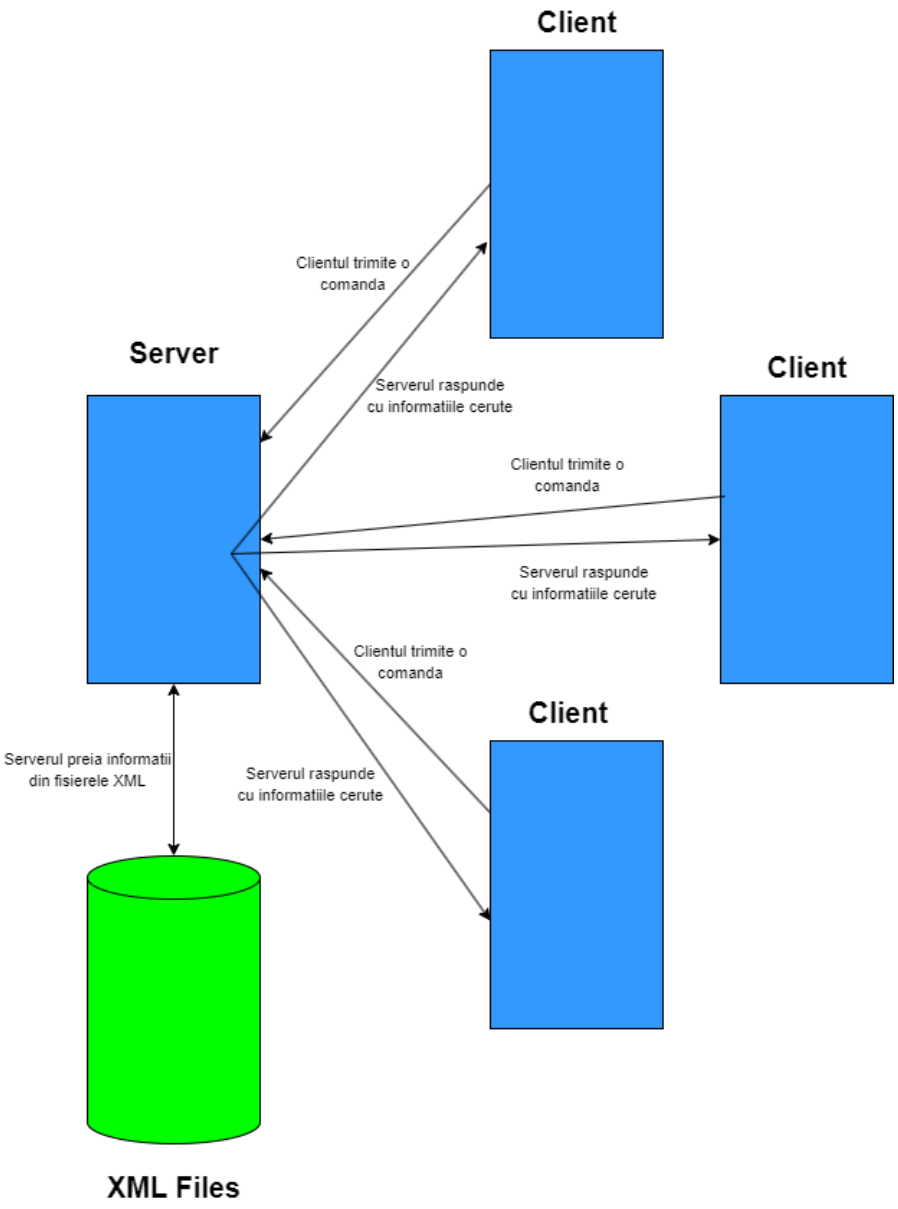

Figure 1: Structura programului

3 Arhitectura aplicaţiei

După cum se observă în Figura 1, structura programului este formată dintr-un server, client(clienţi) şi fişier XML. Mai sus am menţionat că am ales protocolul TCP, deci vom avea modelul client/server TCP şi vor putea rula mai mulţi clienţi în paralel aplicaţia. Un client se va conecta la server şi va introduce numarul comenzii dintre cele trimise sub forma unui string de catre server. Dupa alegerea comenzii, i se va cere sa introduca informaţii cu privire la rută sau numărul unui tren, precum şi data de plecare sau modificari in functie de comanda aleasa. Serverul compară datele primite de la client cu cele care se află în fişierul XML, pe care serverul odată pornit le-a citit şi oferă înapoi un răspuns clientului cu mersul trenurilor. De asemenea, clientul poate trimite ulterior informaţii cu privire la întârzieri. În această situaţie are loc actualizarea fişierului XML, cu noile informaţii prelucrate şi trimise de către server. În continuare server-ul va răspunde la cererile clientului până când acesta se deconectează

- **Clientii** vor afișa o listă de opțiuni primite de la server, din care vor alege comanda dorită.
- **Server-ul** Server-ul va funcționa ca un controler, preluând cereri sub formă de comenzi, mai exact numarul comenzii, de la clienți și efectuând interogări într-o bază de date XML pentru a furniza informații utile călătorilor.
- **Baza de date XML** va conține informații precum numărul trenului, ID-ul stației de plecare și stația de sosire. Server-ul va prelua ulterior aaceste date și le va distribui către clienții corespunzători.

4 Detalii de implementare

4.1

În cadrul aplicației voi folosi thread-uri (firuri de execuție) pentru a eficientiza execuția programelor. Acestea, spre deosebire de procesele copil create de *fork()*, partajează adresele de memorie ale procesului principal, procesele fiind *siblings* în loc de procese părinte și procese copil. Acestea prezintă o serie de avantaje:

- Crearea și eliminarea unui fir de execuție este mai puțin costisitoare din punct de vedere al timpului și resurselor decât în cazul proceselor create de *fork()*
- Context switch-ul este foarte rapid în cazul thread-urilor, deoarece nu e necesară comutarea adreselor de memorie.
- Comunicarea între thread-uri este mai eficientă din punct de vedere al resurselor utilizate precum și al vitezei de comunicare.

```
12   td = (struct thData *)malloc(sizeof(struct thData));
13   td->idThread = i++;
14   td->cl = client;
15
16   pthread_create(&th[i], NULL, &treat, td);
```

Command Design Pattern separa cererile de obiectul care le execută.Clasele InformationCommand, ArrivalsCommand, DeparturesCommand, ModifyCommand și QuitCommand reprezintă diferite comenzi. Toate acestea extind BaseCommand care definește metoda execute() ce trebuie implementată de fiecare comandă. Invoker: Clasa Invoker care execută comanda efectivă.

```
class BaseCommand
{
public:
  virtual const char *execute() const = 0;
};

class InformationCommand : public BaseCommand
{
public:
  const char *execute() const override
  {
    return "Informatii mersul trenurilor: ...\n"
  }
};

class ArrivalsCommand : public BaseCommand
{
public:
  const char *execute() const override
  {
    return "Informatii sosiri: ...\n";
  }
};
```

```
class Invoker
{
public:
  const char *executeCommand(BaseCommand *command)
  {
    return command->execute();
  }
};
```

Command Queue va fi utilizata ulterior, fiind o coada FIFO First-In, First-Out), care stochează comenzi pentru a fi executate. Aceasta va permite serverului să gestioneze cereri multiple într-o manieră eficientă și organizată.

4.2 Baza de date XML

Pentru a lucra în server cu informaţiile din fişierul XML voi folosi biblioteca Libxml2. Cu ajutorul ei voi implementa următoarele funcţionalităţi: citirea datelor din fişier, modificarea datelor din fişier, ştergerea datelor din fişier.

```
<?xml version="1.0"?>
<program>
    <tren ID="R23">
        <statie>
            <ID>Iasi</ID>
            <sosire> 10:33</sosire>
            <maiDevreme>2</maiDevreme>
            <cuIntarziere>40</cuIntarziere>
            <staionare>5</stationare>
        </statie>
    </tren>
</program>
```

Fişierul XML după codul exemplu de mai sus va fi format din taguri cu staţii care conţin trenuri: informatii cu privire la id-ul trenului, ora la care soseşte, durata de întârziere sau de sosire înainte de ora stabilită, durata de staţionare. Acestea vor fi memorate într-o structură.

4.3 Scenarii de utilizare

Când vine vorba de transportul feroviar oamenii au nevoie în general să ştie cum îşi gestionează timpul, dar şi ce posibilităţi există în călătoria lor. Cele mai semnificative utilizări sunt:

- folosirea aplicaţiei pentru a găsi alternative cauzate de întârzieri tehnice sau de accidente.
- oferirea de rute rapide cu legături într-o marjă de timp
- introducerea corectă a unei posibile întârzieri (de exemplu, un tren nu poate întârzia mai mult de 4 ore, altfel este anulat sau călătorul poate găsi altul care soseşte mai devreme)
- aflarea preţului unui bilet în funcţie de statut: adult/copil/student/pensionar
- dacă este permisă călătoria cu animale de companie/biciclete.

5 Concluzii

Dacă timpul este de partea mea şi reuşesc să implementez toate activităţile din cerinţă, îmbunătățirile pe care le voi aduce aplicației mele sunt:

- Un sistem de login / memorare a istoricului cautărilor, de exemplu dacă cineva face zilnic comuta între Iași și Vaslui, aceste informații să fie deja pregătite pentru acesta, nemaifiind nevoie de procesarea datelor.
- Un sistem de sugestie inteligentă a unor trasee alternative pentru utilizator, în cazul în care acesta ratează o anumită cursă.
- Sistem de achiziționare online a biletelor de tren, astfel reducând nevoia de personal la ghiseele gărilor.
- O funcţionalitate legată de timpul la care un călător trebuie să ajungă în staţie în funcţie de parcursul trenului în timp real.

References

[1] Lenuța Alboaie, https://profs.info.uaic.ro/ computernetworks/

[2] https://www.w3schools.com/xml/

[3] http://xmlsoft.org/html/

[4] https://doc.qt.io/

[5] https://www.cfrcalatori.ro/

[6] https://www.infofer.ro/index.php/ro/

[7] https://www.geeksforgeeks.org/command-pattern-c-design-patterns/