Mersul Trenurilor

Grosu Iosif

Facultatea de Informatică, grupa 2B3

1 Introducere

Transportul feroviar, preferat de mulți studenți pentru accesibilitatea sa financiară și rapiditate, întâmpină dificultăți legate de lipsa informațiilor în timp real. O soluție eficientă ar consta în dezvoltarea unei aplicații client-server. Serverul ar oferi informații actualizate despre mersul trenurilor la cererea utilizatorilor, permitând, în același timp, transmiterea către server a informațiilor despre întârzieri sau estimări de sosire din partea anumitor clienți. Astfel, călătorii ar avea acces în orice moment la date precise și actualizate privind trenurile.

2 Tehnologii utilizate

2.1 TCP (Transmission Control Protocol)

Transmiterea corectă și fără erori a informațiilor între client și server este esențială pentru buna funcționare a aplicației. Orice eroare în acest proces poate afecta negativ unul sau mai mulți clienți, conducând la situații precum primirea de informații eronate sau modificarea incorectă a datelor. Pentru a evita aceste probleme la nivelul transportului de date, am ales să implementăm **TCP**, un protocol orientat pe conexiune. Acesta oferă facilități precum transferul de date continuu, fiabilitate prin numere de ordine și confirmări, controlul fluxului, multiplexarea și gestionarea conexiunilor. Astfel, TCP ne asigură transmiterea corectă și ordonată a informațiilor, precum și un mecanism robust de identificare și gestionare a erorilor. De asemenea, facilitățile oferite de TCP permit dezvoltarea unui server concurent, capabil să servească simultan multiple cereri de la diferiți clienți.

2.2 Socket BSD (Berkeley System Distribution)

După alegerea protocolului de transport, următorul pas crucial este selectarea tipului de socket pentru facilitarea comunicării între procese. Socket-ul BSD (Berkeley Software Distribution) este alegerea ideală, fiind un descriptor de fișier care oferă o interfață familiară pentru manipularea datelor de intrare și ieșire în cadrul protocolului domeniului Internet, folosind TCP/IP.

Alegerea socket-ului BSD se justifică prin răspândirea sa în contextul TCP/IP, pe care îl va utiliza și server-ul nostru. În procesul de inițializare, vom specifica domeniul ca AF_INET și tipul ca SOCK_STREAM, asigurând astfel compatibilitate optimă și eficiență în implementarea comunicației în aplicația noastră.

2.3 Fișiere XML

Pentru a executa anumite comenzi, în special cele care implică modificarea fișierului XML, clientul trebuie să se autentifice, deoarece un client neautentificat nu are drepturi de modificare asupra fișierului. Numele de utilizator al clientului va fi stocat într-un fișier de configurare, denumit *config.txt*.

Pentru gestionarea datelor despre trenuri, vom utiliza fișiere XML, deoarece oferă un mecanism accesibil pentru aplicație de a accesa informațiile. Citirea și modificarea datelor din fișierele XML va fi realizată folosind header-ul *tinyxml2*, care furnizează un API pentru manipularea eficientă a datelor XML. Un fragment din fișierul XML ar putea arăta astfel:

```xml
<?xml version="1.0"?>
<trainList>
    <train>
        <name>Blue Arrow</name>
        <Id>CFR001</Id>
        <trainFreeSeats>21</trainFreeSeats>
        <date>09.12.2023</date>
        <stationList>
            <station>
                <cityName>Iasi</cityName>
                <arrivalTime>Start</arrivalTime>
                <arrivalDelay>0</arrivalDelay>
                <departureTime>12:00</departureTime>
                <departureDelay>0</departureDelay>
            </station>
            <station>
                <cityName>Bacau</cityName>
                <arrivalTime>11:20</arrivalTime>
                <arrivalDelay>-100</arrivalDelay>
                <departureTime>11:00</departureTime>
                <departureDelay>-140</departureDelay>
            </station>
            <station>
                <cityName>Vaslui</cityName>
                <arrivalTime>11:40</arrivalTime>
                <arrivalDelay>-160</arrivalDelay>
                <departureTime>Finish</departureTime>
                <departureDelay>0</departureDelay>
            </station>
        </stationList>
    </train>
    <train>
        <name>Red arrow</name>
        <Id>CFR002</Id>
        <trainFreeSeats>15</trainFreeSeats>
        <date>09.12.2023</date>
        <stationList>
```

2.4 Fire de execuție (thread-uri)

Pentru un server TCP concurent, vom utiliza fire de execuție (*threads*) pentru a servi clienții. Această opțiune este mai rapidă decât *fork()* și *select()*, oferind un mod eficient de gestionare a mai multor conexiuni simultane. Thread-ul principal va fi blocat la *accept()*, iar la fiecare conexiune acceptată, se va crea un nou thread dedicat pentru a gestiona respectivul client, folosind *pthread.h*.

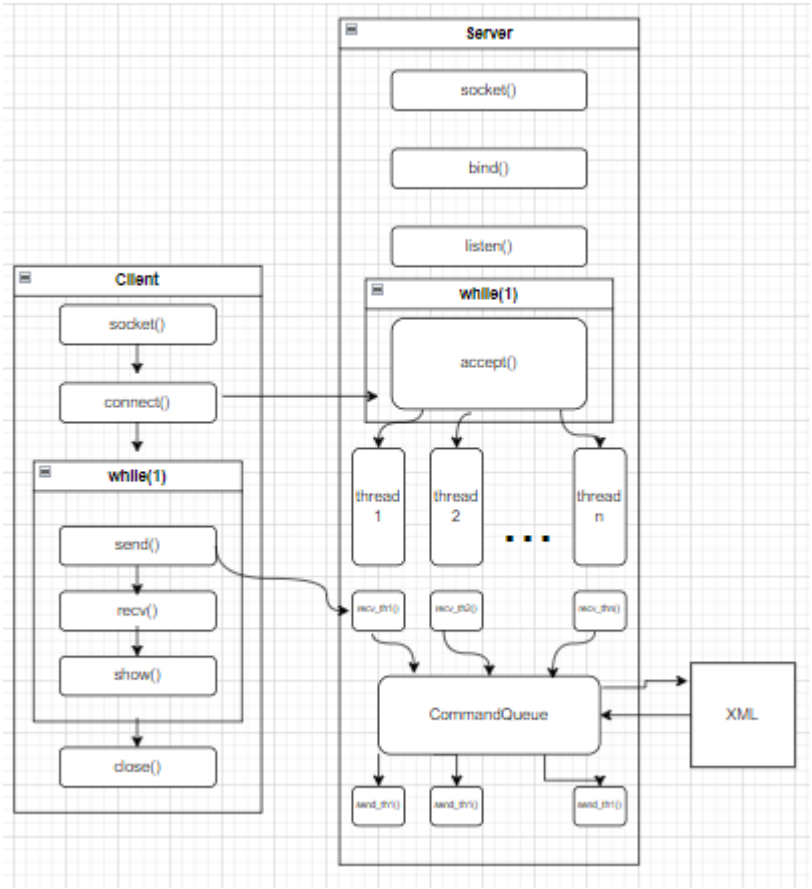
3 Arhitectura aplicației

3.1 Modelul client-server

Modelul client-server distribuie sarcini între un server (furnizor de servicii) și clienți (solicitatori de resurse). Proiectul constă în două aplicații separate, client și server, comunicând prin socket-uri și protocolul TCP. Clientul inițiază conexiunea și trimite comenzi către server, așteptând apoi răspunsul. Serverul procesează comenzile concurent, folosind thread-uri, și returnează rezultatele către clienți.

3.2 Diagrama aplicației

În figura de mai jos este exemplificat, printr-o diagrama, modul în care funcționează și comunică cele două aplicații.



4 Detalii de implementare

4.1 Descrierea protocolului

Clientul este format dintr-un proces care citește de la tastatură o comandă ce aparține protocolului creat și o trimite la server prin intermediul socket-ului. După, se așteaptă ca server-ul să termine procesarea și să trimită rezultatul înapoi. Odată primit mesajul, clientul afișează pe ecran informațiile primite.

Protocolul creat poate fi împărțit în două categorii diferite: comenzi client-public și comenzi client-privat.

Protocolul care implementează comenzile client-public este următorul:

- **Mersul trenurilor în ziua curentă**:*get_trains_info*
- **Detalii despre sosiri**: *getarrivalinfo* sau *getarrivalinfo oraș*
- **Detalii despre plecări**: *getdepartureinfo* sau *getdepartureinfo oraș*
- **Logare**: *login username*
- **Închidere**: *close*

Protocolul client-privat, care poate fi accesat doar dacă clientul este conectat, are următoarele comenzi:

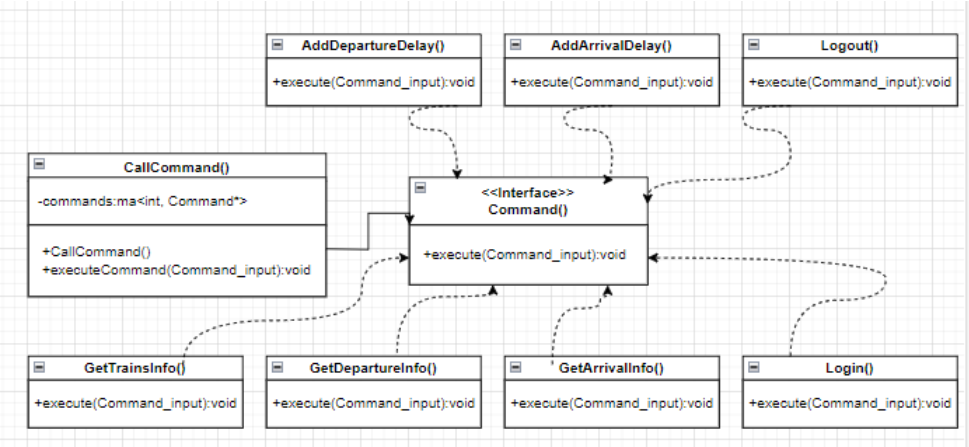
- **Adaugă întârziere la sosire**: *addarrivaldelay idtren statie delay*
- **Adaugă întârziere la plecare**: *adddeparturedelay idtren statie delay*
- **Delogare**: *logout*

Observații:

1. Utilizatorii conectați în modul client-public nu pot accesa comenzile din modul client-privat, în timp ce, clientul conectat în modul client-privat poate accesa toate comenzile.
2. Pentru comanda: *add_arrival_delay id_tren statie delay* si *add_departure_delay id_tren statie delay*, delay-ul poate fi orice număr întreg și are următoarea semnificație:
   - delay < 0: trenul ajunge mai devreme cu *delay* minute;
   - delay $= 0$: trenul ajunge conform cu planul;
   - delay > 0: trenul ajunge cu *delay* minute întârziere;

4.2 Implementarea Command Design Pattern-ului

Pentru a implementa comenzile din protocol vom respecta command design pattern-ul ce este exemplificat în diagrama UML de mai jos:

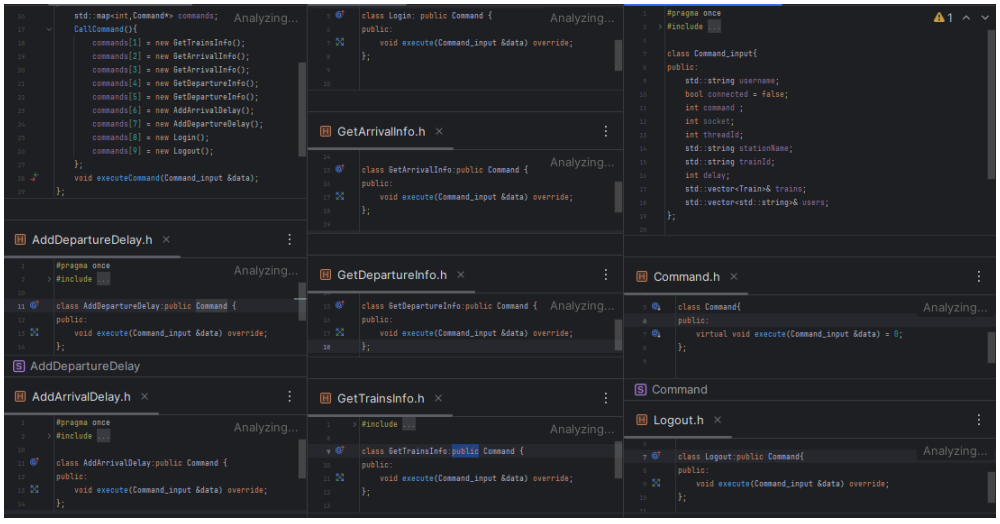


Clasa *Command* este pur virtuală și din ea vom deriva toate clasele necesare pentru implementarea completă a protocolului: *Gettrainsinfo*, *Getarrivalinfo*, *Getdepartureinfo*, *Login*... . Clasa *CallCommand* conține un *map* commands ce va reține indicele comenzii și un pointer către clasa care implementează comanda respectivă.

Structura *command_input* reține următoarele date:

- *socket* - descriptorul socket-ul clientului care execută comanda;
- *command* - comanda apelată;
- *username*, *id*, *station*, *delay* - date necesare pentru executarea corectă a anumitor comenzi;

Implementarea în limbajul C/C++ a diagramei este următoarea:



5 Concluzii

În momentul de față al dezvoltării aplicației, există două probleme destul de mari ce pot crea difucultăți în utilizarea aplicației de către clienți.

Interfața utilizatorului (UI): Utilizarea comenzilor prin terminal poate fi dificilă pentru clienții neobișnuiți cu această interacțiune. O îmbunătățire ar consta în implementarea unei interfețe grafice, permițând utilizatorilor să interacționeze cu aplicația prin butoane și formulare, facilitând astfel accesul la informații despre trenuri.

Coada de comenzi: Actuala implementare a cozi de comenzi prezintă probleme atunci când doi clienți interacționează simultan cu același tren. Prioritizarea comenzilor de modificare a datelor poate duce la înfometarea celor de citire. O alternativă este reținerea ultimelor cinci comenzi de citire pentru fiecare client într-o zi, iar în cazul modificării informațiilor interogate anterior, clientul primește o notificare cu detaliile modificărilor.

6 Bibliografie

1. Cursurile de la adresa: https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
   - Curs 2 - Arhitecturi de rețea
   - Curs 4 - Nivelul Transport
   - Curs 5 - Programare în rețea I
   - Curs 6 - Programare în rețea II
   - Curs 7 - Programare în rețea III
2. https://app.diagrams.net/
3. https://ro.wikipedia.org/wiki/Client-server
4. https://en.wikipedia.org/wiki/Command_pattern
5. https://www.tutorialspoint.com/design_pattern/command_pattern.htm
6. https://leethomason.github.io/tinyxml2/classtinyxml2_1_1_x_m_l_document.html
7. https://en.wikipedia.org/wiki/Command_queue