Mersul Trenurilor

Arseni Costel-Ionuț

Ianuarie 2024

1 Introducere

1.1 Viziune generală

Proiectul "Mersul Trenurilor" are ca scop dezvoltarea unei aplicații server-client pentru furnizarea centralizată a informațiilor despre mersul trenurilor. Aplicația permite accesarea informațiilor legate de rutele trenurilor, orele de plecare și sosire, eventualele întârzieri, estimările de sosire care iau în calcul întârzierile, precum și informațiile despre stațiile intermediare.

Un aspect esențial al aplicației este actualizarea în timp real a datelor, precum statusul plecărilor, sosirilor și întârzierilor. De asemenea, aplicația oferă utilizatorilor posibilitatea de a adăuga noi rute, de a modifica informațiile existente și de a interoga diferite detalii despre trenuri, stații și întârzieri.

2 Tehnologii aplicate

2.1 Protocolul TCP (Transmission Control Protocol)

Pentru implementarea comunicării server-client în cadrul proiectului "Mersul Trenurilor", s-a optat pentru protocolul TCP datorită fiabilității ridicate pe care acesta o oferă în gestionarea datelor critice. Fiind un protocol orientat pe conexiune, TCP garantează livrarea tuturor datelor fără erori, asigurând că acestea ajung la destinație în aceeași ordine în care au fost trimise. Acest aspect este esențial pentru informații precum orele de plecare, sosire și întârzieri.

Spre deosebire de UDP, care nu verifică dacă pachetele au fost pierdute sau livrate într-o ordine greșită, TCP dispune de mecanisme integrate pentru

detectarea erorilor și retransmiterea automată a datelor pierdute, eliminând riscul compromiterii informațiilor. De asemenea, TCP gestionează stabilitatea conexiunii și păstrează integritatea datelor, ceea ce este crucial în acest context, mai ales pentru protejarea fișierelor XML utilizate de server pentru stocarea informațiilor, evitând astfel modificări neașteptate sau pierderi de date.

Întrucât aplicația prioritizează corectitudinea și consistența informațiilor, alegerea TCP este justificată prin nevoia de fiabilitate, chiar dacă presupune un consum mai mare de resurse în comparație cu UDP.

2.2 Gestionarea conexiunilor multiple cu thread-uri

În implementarea actuală a serverului pentru proiectul "Mersul Trenurilor", gestionarea conexiunilor multiple se realizează prin utilizarea thread-urilor, o abordare care oferă flexibilitate și claritate în tratarea interacțiunilor cu clienții. Fiecare client conectat este asociat cu un thread dedicat, ceea ce permite procesarea cererilor acestuia în mod independent de ceilalți clienți.

Codul serverului demonstrează această metodologie prin următoarele etape:

1. Crearea unui socket de ascultare: Serverul inițializează un socket folosind familia de adrese IPv4 (AF_INET) și tipul de socket SOCK_STREAM (pentru conexiuni TCP).

2. Ascultarea conexiunilor: După legarea (bind) socket-ului la portul specificat, serverul intră în modul de ascultare, pregătit să accepte conexiuni de la clienți.

3. Acceptarea conexiunilor: La fiecare conexiune nouă, un socket client este creat folosind funcția accept(), iar un thread dedicat este lansat pentru procesarea cererilor clientului respectiv.

Fiecare thread este separat de execuția principală și prelucrează cererile clientului, ceea ce simplifică procesarea paralelă.

2.3 Limbajul de programare

Pentru acest proiect, am ales limbajul de programare C++, care oferă un echilibru excelent între performanță și ușurință în utilizare, având în vedere beneficiile bibliotecii standard C++ (STL). C++ este bine-cunoscut pentru

performanța sa superioară și gestionarea eficientă a resurselor, dar, în același timp, oferă o gamă largă de facilități care ajută la dezvoltarea rapidă și eficientă a aplicațiilor.

Unul dintre principalele avantaje ale C++ este biblioteca standard (STL), care include o serie de clase și funcționalități predefinite care simplifică multe sarcini de programare. Printre aceste beneficii se numără:

- Manipularea datelor: C++ oferă tipuri de date puternice, cum ar fi `std::string`, care simplifică lucrul cu șiruri de caractere. De asemenea, `std::vector` este o structură de date eficientă care permite gestionarea colecțiilor de date fără a te preocupa de gestionarea manuală a memoriei.

Astfel, C++ aduce o combinație de performanță, control asupra resurselor și funcționalități moderne care permit dezvoltarea rapidă și eficientă a aplicațiilor, precum "Mersul Trenurilor".

2.4 Gestionarea fișierelor XML

Fișierele XML sunt utilizate în cadrul proiectului "Mersul Trenurilor" pentru gestionarea datelor, datorită avantajelor pe care acest format le oferă în stocarea și manipularea informațiilor structurate. Printre aceste avantaje se numără faptul că XML este un format standardizat, independent de platformă și larg acceptat, ceea ce asigură portabilitatea datelor și facilitează integrarea cu alte sisteme.

În contextul proiectului, fișierele XML sunt ideale pentru stocarea informațiilor despre trenuri, precum rutele, orele de plecare și sosire sau întârzierile. Acestea permit organizarea datelor într-o structură ierarhică clară, ceea ce face ca informațiile să fie ușor de înțeles și de prelucrat. Datorită acestei structuri, XML facilitează extragerea, actualizarea și validarea datelor, făcându-l un instrument valoros pentru gestionarea eficientă a datelor proiectului.

În plus, utilizarea fișierelor XML permite compatibilitatea cu o gamă largă de tehnologii și instrumente de procesare, inclusiv limbaje precum Java, Python și C++, care dispun de biblioteci dedicate pentru manipularea XML. Acest lucru adaugă un nivel suplimentar de flexibilitate în implementarea soluțiilor software.

Astfel, XML se dovedește a fi un format fiabil și eficient pentru nevoile proiectului "Mersul Trenurilor", contribuind la gestionarea structurată și portabilă a informațiilor despre trenuri.

3 Structura aplicației

Arhitectura aplicației utilizează un model de comunicare între server și client bazat pe crearea unui thread separat pentru fiecare client conectat. Această metodă permite procesarea concurentă a cererilor, asigurând un timp de răspuns rapid și gestionarea eficientă a resurselor hardware.

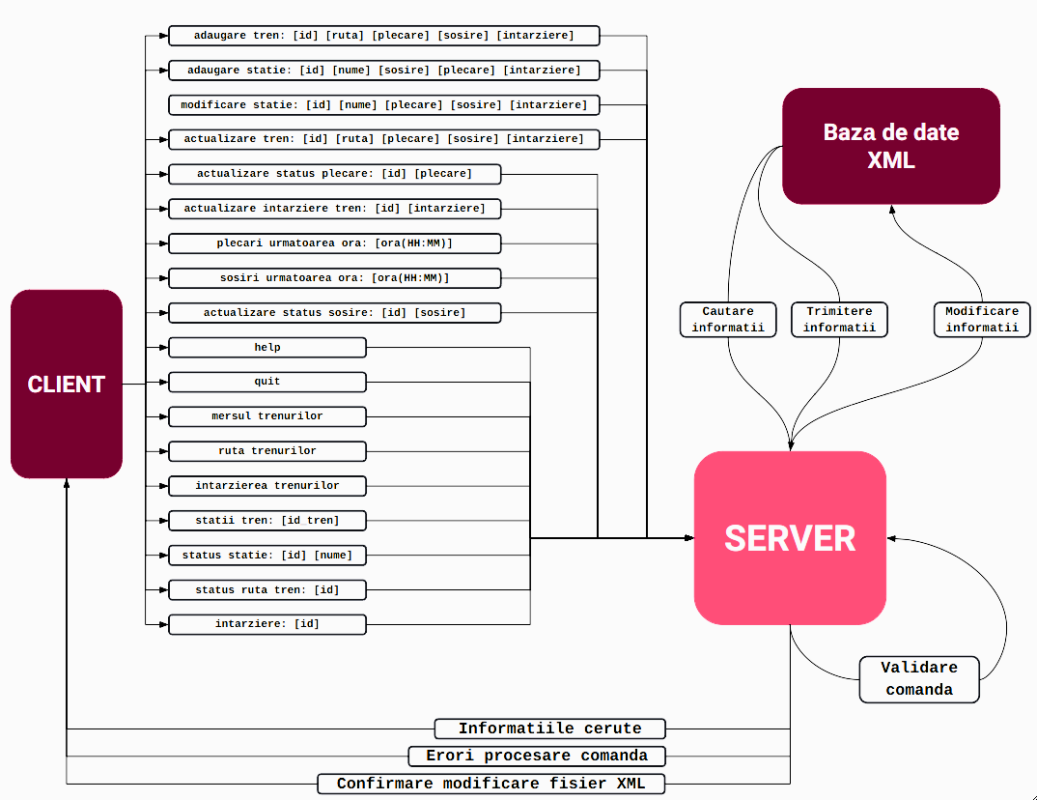

Figure 1:

3.1 Configurarea serverului

Serverul este configurat pentru a accepta conexiuni multiple de la clienți, utilizând următorii pași:

1. Crearea socket-ului de ascultare folosind funcția `socket()`.

2. Setarea opțiunii `SO_REUSEADDR` cu funcția `setsockopt()` pentru a permite reutilizarea adresei.

3. Legarea socket-ului la o adresă IP și port prin funcția `bind()`.

4. Pornirea ascultării conexiunilor cu `listen()`.

3.2 Crearea unui thread pentru fiecare client

Pentru fiecare conexiune nouă, serverul creează un thread separat, asigurând izolare și concurență între procesele fiecărui client. Funcția `thread_client` gestionează interacțiunile cu un client specific, incluzând citirea comenzilor, procesarea acestora și trimiterea răspunsurilor.

3.3 Gestionarea comenzilor și utilizarea mutex-urilor

Funcția `lansare_comenzi` este utilizată pentru a gestiona comenzile trimise de client. Aceasta blochează accesul concurent la resursele partajate utilizând `pthread_mutex_lock()` și `pthread_mutex_unlock()` pentru a preveni conflictele.

3.4 Avantajele utilizării unui thread separat

1. Concurență: Fiecare client este gestionat independent de un thread, fără a afecta alte conexiuni.

2. Performanță ridicată: Comunicarea asigură răspunsuri rapide datorită procesării independente.

3. Eficiență în utilizarea resurselor: Modelul de thread-uri minimizează consumul de resurse, evitând necesitatea de procese separate.

4. Scalabilitate: Serverul poate gestiona simultan un număr mare de conexiuni.

Această arhitectură permite serverului să rămână scalabil, să răspundă prompt solicitărilor și să gestioneze în mod eficient mai mulți clienți conectați în același timp.

3.5 Verificarea existenței fișierului XML

Funcția `file_exists` verifică dacă un fișier XML specificat există pe disc. Aceasta folosește funcția `stat` din biblioteca standard pentru a determina dacă fișierul este accesibil. Dacă fișierul există, funcția returnează true; în caz contrar, false.

3.6 Crearea fișierului XML

Funcția `create_xml_file` creează un nou fișier XML utilizând biblioteca `libxml2`. Fișierul este denumit `trenuri.xml` și are următoarea structură de bază:

- Rădăcina: `<trenuri>`.
- Subcategoria implicită: `<tren>`, care este adăugată automat sub rădăcina `<trenuri>`.
- Opțional: În interiorul subcategoriei `<tren>`, se poate adăuga un nod `<statie>` cu conținut, de exemplu.

3.7 Verificarea și crearea fișierului XML

Funcția `verifica_si_creeaza_fisier` verifică dacă fișierul XML există folosind funcția `file_exists`. Dacă fișierul nu există, acesta este creat automat folosind funcția `create_xml_file`.

- În cazul în care fișierul există deja, programul trece mai departe la prelucrarea comenzilor, modificarea bazei de date sau afișarea informațiilor.

3.8 Verificarea existenței trenurilor și stațiilor

- Funcția `id_existent` verifică dacă un tren cu un anumit `id_tren` există în documentul XML. Parcurge nodurile de tip `<tren>` pentru a găsi atributul `id` care corespunde.
- Funcția `statie_existent` verifică dacă o stație cu un anumit `nume_statie` există în lista de stații ale unui tren specificat. Aceasta parcurge nodurile `<statie>` și verifică nodurile copil `<nume>`.

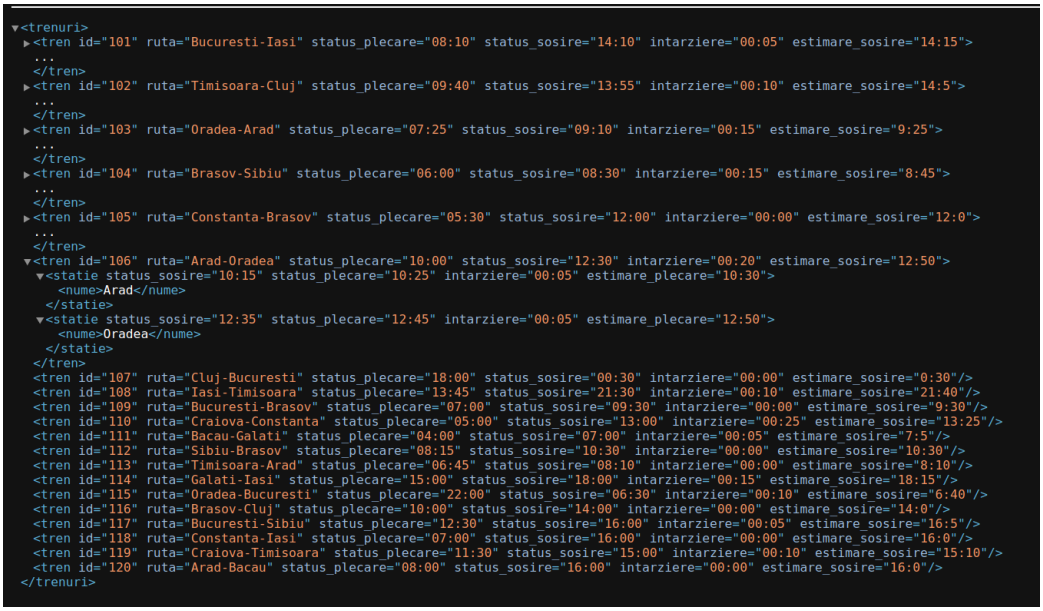

Figure 2:

4 Aspecte de implementare

Unul dintre aspectele esențiale ale implementării este parsarea comenzilor trimise de client către server. Acest proces este fundamental pentru ca serverul să poată înțelege și executa corect instrucțiunile primite. Comenzile sunt de obicei trimise sub formă de șiruri de caractere, iar rolul serverului este să le analizeze și să le interpreteze conform unui set prestabilit de reguli.

4.1 Parsarea comenzilor

Parsarea comenzilor presupune divizarea unui șir de caractere într-un format care să poată fi interpretat corect de aplicație. Funcția principală care realizează acest lucru este `validare_comanda()`, care analizează comanda primită de la client și o compară cu comenzile predefinite. De exemplu, comenzile pot include operațiuni de tipul `help`, `quit`, sau comenzi mai complexe care necesită argumente adiționale, precum `adaugare`.

De exemplu, în cazul unei comenzi de tipul `adaugare tren:<id> <argumente>`, funcția de parsare va separa argumentul de comanda propriu-zisă și va valida dacă `<id>` este un număr valid care poate fi asociat unui tren existent sau

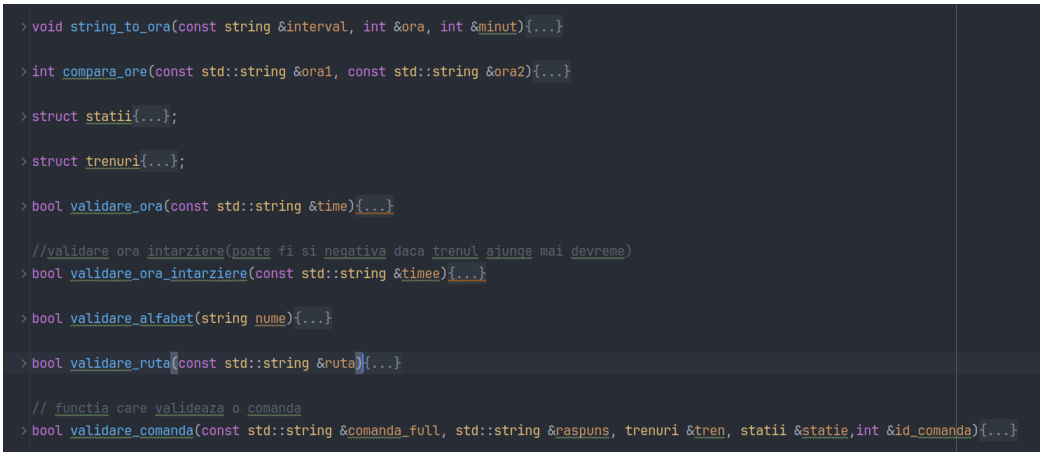

Figure 3:

unei rute corecte. În cazul în care formatul nu este corect sau argumentele sunt invalide, serverul va trimite un mesaj de eroare clientului, îndrumându-l să utilizeze comanda `help` pentru a vizualiza comenzile valide.

4.2 Mesaje de feedback

Pe lângă procesul de parsare, validarea este însoțită de mesaje de feedback clare pentru utilizator. Dacă o comandă este validă, serverul răspunde cu un mesaj de confirmare, indicând faptul că cerința a fost îndeplinită. În caz contrar, clientul primește un mesaj de eroare care detaliază problema și sugerează corectarea comenzii. De exemplu, pentru comanda incorectă `adaugare tren: 1234a` , serverul va trimite un mesaj de eroare care specifică faptul că nu sunt suficiente argumente. Această abordare ajută la asigurarea unui flux corect de date și la minimizarea erorilor de comunicare între client și server, îmbunătățind experiența utilizatorului prin mesaje clare și precise.

5 Comenzile acceptate

Aplicația suportă următoarele 18 comenzi valide pentru gestionarea și interogarea datelor, plus comanda ”Trimite”, de trimitere a comenzilor la

server:

5.1 adaugare tren:

- `[id_tren] [ruta] [status_plecare] [status_sosire] [intârziere]`

Adaugă un nou tren cu ruta și informațiile specifice.

5.2 adaugare statie:

- `[id_tren] [nume_statie] [status_sosire] [status_plecare] [intârziere]`

Adaugă o stație pentru un tren existent.

5.3 modificare statie:

- `[id_tren] [nume_statie] [status_sosire] [status_plecare] [intârziere]`

Modifică informațiile pentru o stație existentă.

5.4 status ruta tren:

- `[id_tren]`

Afișează statusul complet al rutei trenului.

5.5 status statie:

- `[id_tren] [nume_statie]`

Afișează informațiile despre o stație specifică.

5.6 actualizare tren:

- `[id_tren] [ruta] [status_plecare] [status_sosire] [intârziere]`

Actualizează toate informațiile unui tren.

5.7 actualizare status plecare:

- `[id_tren] [status_plecare_nou]`

Actualizează ora de plecare pentru un tren.

5.8 actualizare status sosire:

- `[id_tren] [status_sosire_nou]`

Actualizează ora de sosire pentru un tren.

5.9 actualizare intarziere tren:

- `[id_tren] [intârziere]`

Actualizează întârzierea asociată unui tren.

5.10 plecare urmatoarea ora:

- `[ora (HH:MM)]`

Afișează trenurile care au plecări programate în următoarea oră.

5.11 sosire urmatoarea ora:

- `[ora (HH:MM)]`

Afișează trenurile care au sosiri programate în următoarea oră.

5.12 intarzierea trenurilor

Listează toate trenurile care au întârzieri.

5.13 intarziere:

- `[id_tren]`

Afișează întârzierile pentru un tren specific.

5.14 mersul trenurilor

Afișează informații despre toate trenurile în circulație.

5.15 ruta trenurilor

Afișează rutele tuturor trenurilor.

5.16 statii tren:

- `[id_tren]`

Afișează lista stațiilor pentru un tren specific.

5.17

- `help`: Afișează lista tuturor comenzilor disponibile.
- `quit`: Închide conexiunea client-server.

5.18 Observații privind validitatea comenzilor

- `[ruta]`: `[STATIE1]-[STATIE2]`, unde fiecare stație este un șir de caractere format doar din litere.
- `[id_tren]`, `[id_statie]`: Șiruri de caractere unice pentru identificare.
- `[status_plecare]`, `[status_sosire]`: Formate de tip (HH:MM), unde:
  - HH: Număr întreg din intervalul 0-23.
  - MM: Număr întreg din intervalul 0-59.
- `[intârziere]`: Poate avea formatele (HH:MM) sau -(HH:MM) pentru întârzieri negative.
- `[nume_statie]`: Șir de caractere format doar din litere.
- `Trimite`: Utilizatorul trebuie să trimită comanda `"Trimite"` pentru ca serverul să înceapă să primească și să proceseze comenzile.

6 Concluzii

Extinderea funcționalităților aplicației ar putea include planificarea călătoriilor și rezervarea biletelor. Acestea ar contribui la crearea unei aplicații mai complexe și utile. Integrarea unor sisteme externe, cum ar fi notificările în timp real despre întârzieri sau informațiile meteorologice, ar îmbunătăți considerabil performanța și utilitatea aplicației pentru utilizatori.

În cadrul aplicației, am implementat funcții care permit gestionarea fișierelor XML. Aceste funcționalități includ verificarea existenței fișierelor XML, crearea lor automată atunci când lipsesc și operațiuni precum verificarea existenței trenurilor și a stațiilor asociate, precum și de modificare, actualizare și afișare a bazei de date.

Prin urmare, aplicația dispune acum de o bază solidă pentru manipularea fișierelor XML, integrând eficient relația dintre server și client și permițând extinderea ușoară a funcționalităților viitoare.

7 Referințe Bibliografice

- libxml2 Documentation. (n.d.). Retrived from https://xmlsoft.org/
- https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines
- https://en.wikipedia.org/wiki/TransmissionControlProtocol
- cppreference.com. (n.d.). C++ reference. Retrived January 9, 2025, from https://en.cppreference.com/w/
- https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php