Raport tehnic - Mersul Trenurilor

Vasilică Alexandru

Facultatea de Informatică Iași

Abstract. Un server ce oferă sau actualizează informații în timp real de la toți clienții înregistrați pentru: mersul trenurilor, status plecări, status sosiri, întârzieri și estimare sosire. Serverul citește datele din fișiere xml si actualizează datele(întârzieri si estimare sosire) la cererea clienților

Keywords: Server TCP concurent · Multithreading · Command design pattern · Command queue · Fișier XML

1 Introducere

1.1 Server

Proiectul constă în crearea unui server TCP concurent ce poate oferi și actualiza informații despre mersul trenurilor în timp real clienților conectați. Datele necesare furnizării informațiilor necesare vor fi procurate dintr-un fișier în format XML. În cadrul proiectului a fost utilizat fișierul publicat de CFR Călători. Funcționalitățile pe care serverul își propune să le pună la dispoziție sunt:

- transmiterea datelor despre sosiri și plecări din stația curentă a clienților în ziua curentă
- oferirea clienților posibilitatea de a raporta posibile întârzieri ale trenurilor sau de a estima sosirea în avans a acestora
- comunicarea datelor despre statutul actual al plecărilor și sosirilor din stația curentă a clienților în următoarea oră(luând în calcul posibilele modificări în program)

1.2 Client

Totodată proiectul va pune la dispoziție un client ce va permite conectarea la server și transmiterea acestuia comenzile specifice, necesare pentru trimiterea și primirea informațiilor, prin intermediul liniei de comandă.

2 Tehnologii Aplicate

2.1 Protocolul TCP

Pentru a asigura transmiterea corectă, fără pierderi de informații dintre server și client, acestea stabilesc o conexiune bidirecțională prin intermediul protocolului TCP, ce permite trimiterea de comenzi de la client la server și recepționarea ulterioară a datelor cerute. Astfel integritatea și ordinea biților transmiși este păstrată. Implementare este efectuată prin intermediul socket-urilor și operațiilor de scriere și citire din limbajul C și API-ul POSIX.

2.2 Api-ul pentru thread-uri Pthread

Thread-urile reprezintă fire de execuție ce permit efectuarea în paralele a diferitelor task-uri.

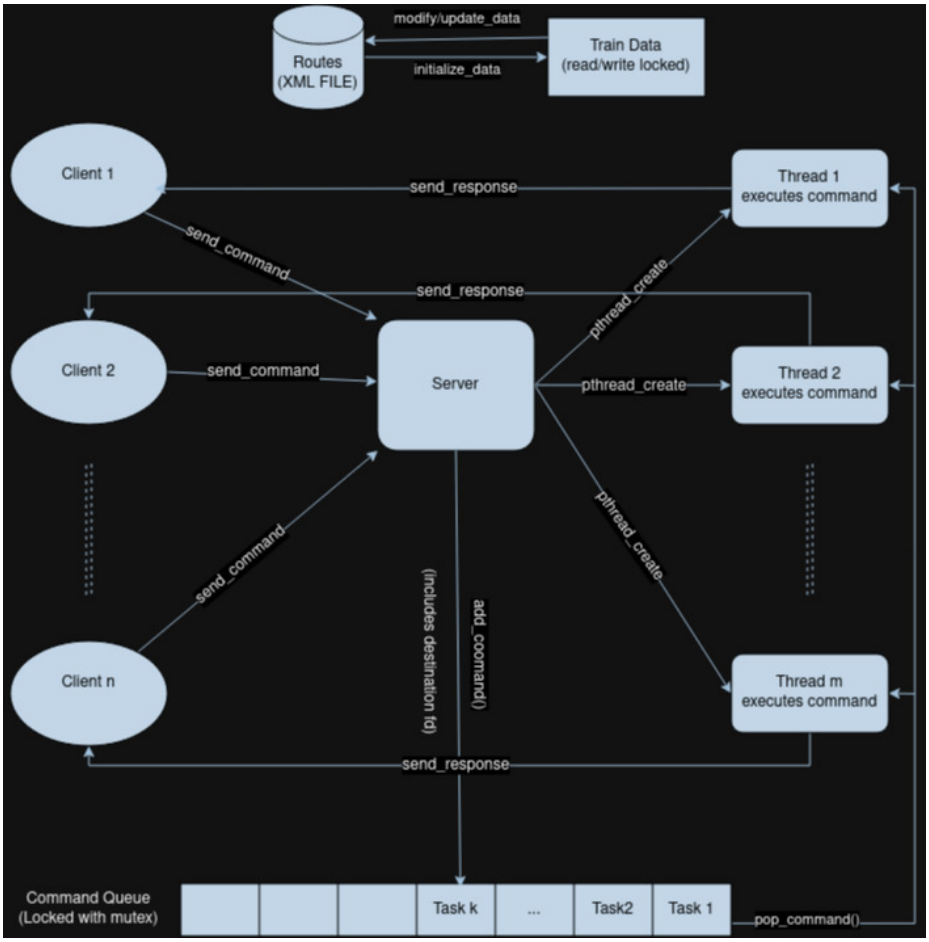
Pthread_mutex este un mutex ce permite sincronizarea diferitelor fire de execuție astfel încât să se evite un posibil data race.

Pthread_condition este o variabilă de condiție ce permite thread-urilor să intre în waiting mode și să nu mai consume resurse până la primirea unui semnal prin intermediul variabilei.

Pthread_rwlock este un lacăt ce permite sincronizarea operațiilor de scriere și citire din diferite thread-uri asupra unui secțiuni critice. Aceasta asigură accesul la secțiunea critică fie unui singur scriitor, fie mai multor cititori.

3 Structura Aplicației

Structura aplicației este prezentată în următoarea diagramă:

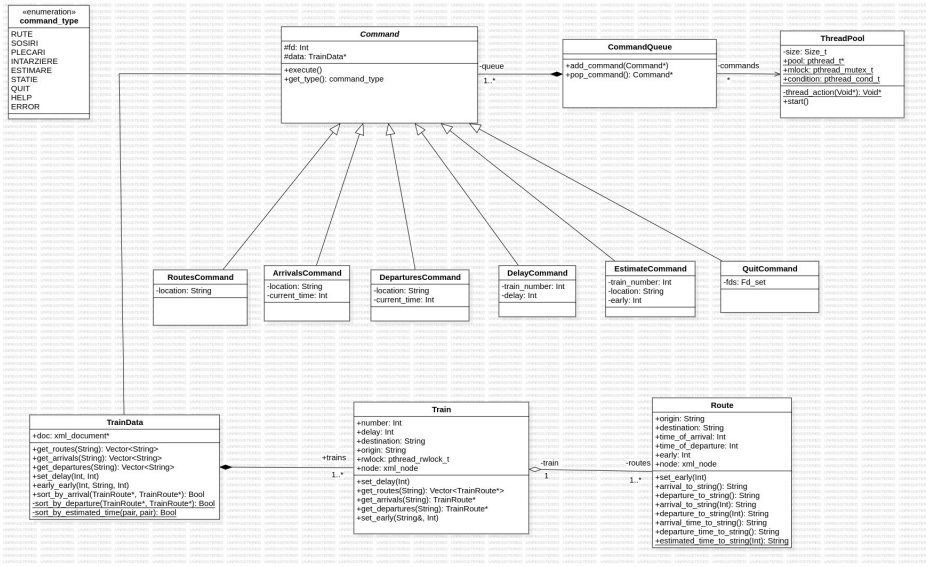


Pentru a permite procesarea comenzilor mai multor clienți în același timp serverul va inițializa un număr de thread-uri, ce așteaptă să primească un task ce trebuie executat. Transmiterea și sincronizarea comenzilor se face prin intermediul unei cozi, blocată printr-un mutex, ce ajută la evitarea situațiilor în care 2 thread-uir diferite preiau același task. Totodată este inițializată o structură ce facilitează obținerea datelor necesare prin parsarea fișierului XML.

Clienții se conectează la server, iar apoi aceștia pot trimite diferite cereri de date prin intermediul comenzilor puse la dispoziție. Serverul monitorizează descriptorii clienților conectați prin select și așteaptă să poată citi cererile. Mesajul clientului este transformat într-un obiect de tipul Command și adăugat în coadă de server. Totodată serverul semnalează threadurilor existența unui nou task printr-o variabilă de condiție.

După execuția comenzii și obținerea sau modificarea datelor necesare, threadul care a preluat task-ul va scrie în socket-ul aferent răspunsul cerut și așteaptă să preia o nouă comandă.

Următoarea diagramă UML exemplifică structura claselor utilizate în server, structurată conform paradigmei Command design pattern:



4 Aspecte de implementare

4.1 Command Pattern

La baza aplicației se află obiectele derivate din clasa abstractă Command. Fiecare comandă are o metodă execute specifică acesteia. În structura fiecărei comenzi se află informațiile necesare pentru obținerea datelor cerute din structura TrainData (locația actuală a clientului, descriptorul unde va fi scris răspunsul, etc.).

Serverul creează comandă corespunzătoare din structura request trimisă de client și le adauga în structura CommanQueue.Comenzile de modificare a datelor (raportarea întârzierilor și estimarea sosirii) vor fi adăugate în capul cozii, astfel încât execuția lor să fie prioritizată, iar cele care doar citesc date sunt adăugate la coadă.

Înainte să accepte clienți serverul parsează fișierul XML și creează structura TrainData care reține datele necesare generării răspunsurilor. Parsarea se face prin intermediul librăriei open-source pugixml și construcția clasei respectă structura fișierelor XML publicate de compania CFR la adresa din bibliografie.

4.2 Sincronizarea thread-urilor

Următoarea secțiune de cod exemplifică metoda utilizată de server pentru primirea de noi comenzi și adăugarea lor în coadă. Este folosită funcția select pentru a îmbunătății eficiența serverului în recepționarea mesajelor mai multor clienți, putând astfel să determine dacă poate citi dintr-un anumit socket păstrat într-un set separat de descriptori.

```
1  //verificarea descriptorilor rezultati din apelul select
2  //accepatrea noilor clienti a fost facuta anterior
3  for (fd = 0; fd <= nfds; fd++)
4          {
5          //citirea din descriptorii pregatiti pentru
               procesare
6              if (fd != sd && FD_ISSET(fd, &readfds))
7              {
8                  struct request req;
9                  int bytes=read(fd, &req, sizeof(struct
                       request));
10
11                 if(bytes) {
12                     //Cazul in care s-a primit un
                           request
13                     //setarea descriptorului unde va fi
                           scris raspunsul
14                 req.fd=fd;
15                 //adaugarea comenzii aferente tipului
                       de request
16                 commandQueue.add_command(get_request(
                       req,*data,actfds));
17                 printf("Added request from descriptor
                       %d to queue\n", fd);
18                 //semnalarea catre threaduri a
                       existentei unei noi comenzi in
                       coada
19                 pthread_cond_signal(&condition);
20                 } else{
21                     //Cazul in clientul a inchis
                           conexiunea
22                 FD_CLR(fd,&actfds);
23                 }
```

Thread-urile execută următoarele instrucțiuni. Mutex-ul mlock asigură accesul exclusiv la extragerea unei comenzi din coadă, iar variabila de condiție condition permite propagarea semnalelor serverului la adăugarea unei noi comenzi în coadă. Fără aceasta thread-urile ar încerca în mod repetat să preia un task și ar consuma resursele procesorului. Prin această metodă, în cazul în care nu există comenzi în coadă, thread-ul va intra în modul de așteptare și eliberează mutex-ul, urmând să se trezească atunci când primește semnalul de la firul principal de execuție.

```
1  //Functia folosita in pthread_create
2   void*ThreadPool::thread_action(void *arg) {
```

```
3   //idx reprezinta numarul de ordine al threadului pasat
        ca argument
4       int idx=*(int *)arg;
5   //intrucat nu facem join, threadurile vor elibera
        resursele in momentul in care isi termina executia
6       pthread_detach(pthread_self());
7       //main loop
8       while(true) {
9       //blocarea cozii de comenzi
10          pthread_mutex_lock(mlock);
11          if (commands.empty())
12          //daca coada e goala threadul doarme pana
                primeste semnalul ca poate prelua o comanda
13              pthread_cond_wait(condition,mlock);
14          command=commands->pop_command();
15          //deblocarea cozii dupa preluarea comenzii
16          pthread_mutex_unlock(mlock);
17          //apelul specific de executie a comenzii si
                trimitere a datelor
18          command->execute();
19      }
20  }
```

4.3 Clientul

La deschiderea aplicației client utilizatorul își introduce stația actuală, care este trimisă odată cu fiecare cerere de date. Când serverul acceptă conexiunea, automat va trimite clientului toate plecările și sosirile din acea stație în ziua respectivă. Ulterior utilizatorul poate introduce următoarele comenzi recunoscute de client:

- rute → afișează toate plecările și sosirile din stația actuală în ziua respectivă
- sosiri → afișează toate sosirile în stația actuală în următoarea oră, luând în calcul posibilele întârzieri sau veniri în avans
- plecări → afișează toate plecările stația actuală în următoarea oră, luând în calcul posibilele întârzieri
- întârziere <nr. tren> <nr. minute> → raportează o întârziere cu un anumit număr de minute față de ora din program a trenului cu numărul dat
- estimare <nr. tren> <nr. minute> → raportează o sosire în avans cu un anumit număr de minute față de ora din program a trenului cu numărul dat în stația actuală a clientului
- stație <nume stație> → modifică stația actuală a clientului cu cea introdusă și trimite automat serverului o cere de afișare a plecările și sosirile din stația actuală în ziua respectivă
- quit → închide conexiunea cu serverul și oprește execuția programului

- help → afișează funcționalitățile aplicației
  (Parametrii formați din mai multe cuvinte vor fi scriși între ghilimele)

După ce primește o comandă de la intrarea standard clientul creează o structură request pe care o trimite serverului, iar popi așteaptă răspunsul. Răspunsul este format din citirea unui număr de linii ce vor fi recepționate, iar ulterior citirea și afișarea fiecărui șir de caractere cu o dimensiune fixată. În cazul în care comanda dată este invalidă, clientul va afișa un mesaj de eroare și va aștepta o nouă comandă.

4.4 Scenarii de utilizare

Întrucât aplicația preia datele din fișiere XML ce respectă formatul utilizat de compania CFR, aplicația poate fi utilizată de posibili călători pentru a se informa în legătură cu programul rutelor feroviare.

5 Concluzii

Câteva îmbunătățiri ce ar putea fii aduse la aplicație ar fi:

- preluarea automată a locației curente a clientului și determinarea celei mai apropiate stații
- implementarea unei interfețe grafice ce ar simplifica accesul la informații
- adăugarea unui mecanism de chaching pentru cererile clienților, reducând astfel numărul de accesări ale datelor și timpul de așteptare pentru un răspuns

6 Referințe Bibliografice

References

1. Setul de date al rutelor CFR utilizat, https://data.gov.ro/dataset/mers-tren-sntfc-cfr-calatori-s-a/resource/1b0b64c9-389d-4a83-b3f0-f8987cabe479
2. Pugixml, https://pugixml.org/, MIT License
3. Librăria Pthread, https://man7.org/linux/man-pages/man7/pthreads.7.html
4. Command Pattern, https://en.wikipedia.org/wiki/Command_pattern
5. Site curs Rețele de calculatoare, https://profs.info.uaic.ro/~computernetworks/
6. Site laborator Rețele de calculatoare, https://sites.google.com/view/fii-lab-retele-de-calculatoare/