Monitorizarea Traficului

Spiridon Roberto-Riccardo

Universitatea Alexandru Ioan Cuza, Facultatea de Informatica
robertospiridon1@gmail.com

1 Introducere

Proiectul *"Monitorizarea Traficului"* este un proiect complex, care gestionează traficul rutier și oferă informații utile șoferilor. Scopul acestei aplicații server-client este de a îmbunătăți semnificativ siguranța rutieră. Ca funcționalități se numără raportarea incidentelor, primirea de informații de către șoferi cu privire la condițiile meteo, evenimentele sportive, prețurile pentru combustibil la stațiile PECO(motorina-benzina),debifarea acestor optiuni daca vrea soferul, dar și viteza recomandată pe o anumită porțiune de drum. O alta functionalitate este **verificarea vitezei la o frecventa de 1 minunt.**

Din cele enumerate mai sus, aceste funcționalități pot aminti de aplicația Google Maps sau mai nou apărută, Waze. Aceasta a fost cumparată de Google dupa 7 ani de la lansare. Aplicația se concentrează pe un server TCP concurent și interacțiunea acestuia cu mai mulți clienți. Regăsiți la capitolele viitoare de ce am ales un TCP concurent.

2 Tehnologii Aplicate

Cum am spus mai sus, vom folosi un TCP concurent, cât și o baza de date.

- **TCP:** Vă voi explica de ce dintre TCP sau UDP am ales TCP. Ne imaginăm ca șoferii și serverul sunt persoane care își trimit scrisori. Ele conțin informații. Să zicem că un șofer trimite o scrisoare catre server pentru a raporta un accident. TCP garantează că scrisoarea, impreună cu informațiile din ea ajung la destinatar fără pierdere. Chiar și in cel mai rău caz, dacă se pierde scrisoarea, acesta o va retrimite la destinatar până ajunge in siguranță. Din celălalt colț, curierul(prin UDP) nu mai intreabă dacă ai primit scrisoarea, el trimite foarte rapid și nu ține cont de integritatea informațiilor din scrisori. Am putea spune ca TCP este un curier grijuliu și UDP un curier grăbit.

- **MySQL:** Baza de date folosită în acest proiect este **MySQL** deoarece este performantă și gestionează un numar mare de date. Este folosită pe scară largă in industrie, folosind limbajul SQL(Structure Query Language). Interacțiunea cu baza de date se va face prin interogări SQL, utilizând biblioteca `mysql/mysql.h`. In această bază de date voi retine informații despre viteze, numele clienților(cu informatii inițiale despre viteza cu care circulă si pe ce stradă), evenimente sportive, raporturi de accidente dar și despre condițiile meteo.

3 Structura Aplicației

După cum am spus și in *"Introducere"*, aplicația este formată din două entități principale: serverul și clienții. Serverul utilizează un protocol TCP concurent, astfel încât mai mulți clienți pot comunica simultan cu serverul. Voi incepe prima dată cu diagrama pentru a vă face o idee despre cum se trimit și se citesc datele, dar și de interacțiunea cu o bază de date.

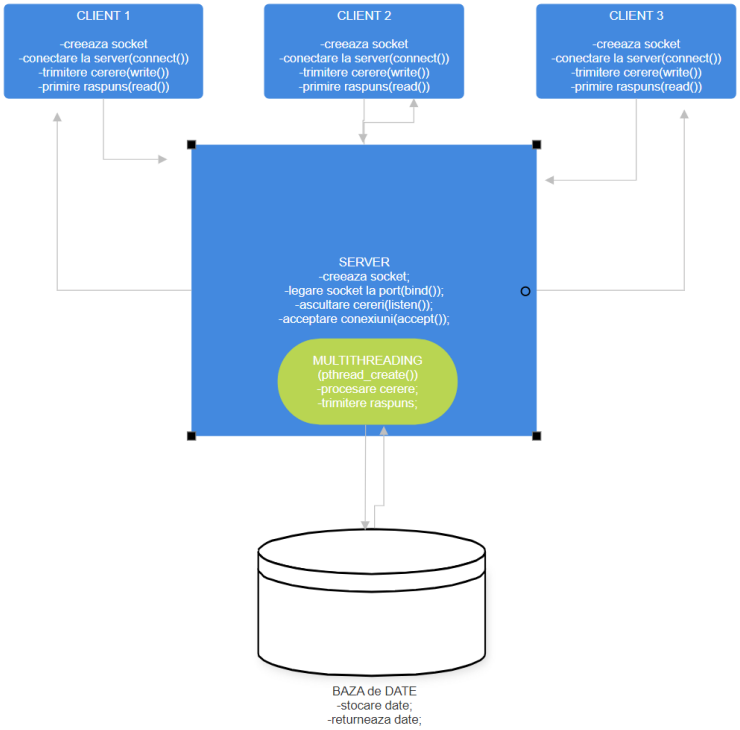

Fig. 1. Diagrama arhitecturală a aplicației

- **Diagrama generală a aplicației**

Serverul creează un socket prin funcția `socket()` și îl leagă de un port folosind `bind()`. Apoi, serverul ascultă conexiunile cu `listen()` și acceptă conexiunile utilizând `accept()`.

Aici intervine **Multithreading-ul**. Ce inseamnă asta? Fiecare cerere a unui client este luată de unul dintre thread-uri ca să fie procesarea paralelă. Deci pentru fiecare conexiune acceptată se foloseste un thread. Am implementat un thread in server pentru a verifica viteza fiecarui sofer la frecventa de 1 minut si ii este afisat fiecarui client pe ecran. Ii v-a afisa daca a depasit limita de viteza de pe strada pe care se afla sau nu. Acest thread este creat in client.

Acum apare **baza noastră de date** unde sunt stocate informații prețioase pentru șoferi: despre vreme, limita de viteză, evenimente sportive, nume utilizator(cu numele străzii si viteza initală) dar și pretul pentru combustibil. Apoi serverul după ce preia aceste informații trimite răspunsuri către clienți. Baza date date MySQL o voi initializa in server si voi avea un fisier separat in care creez baza de date cu tabele si inserari in acestea.(Tabela - Soferi, Optiuni, Strazi).

La rândul lui, **clientul** creează un socket pentru a se conecta la server folosind funcția `connect()`. Odată conectat, clientul poate trimite cereri de informații și poate raporta accidente. Serverul primește aceste cereri, le prelucrează și trimite răspunsurile corespunzătoare către clienți. In client am implementat un thread care afiseaza fiecare raspuns de la server, ca sa nu fie probleme de suprapunere.

Acum, dupa ce v-am explicat mai mult partea teoretică cu privire la funcționalitatea TCP-ului concurent in aplicația noastră, vă voi prezenta o altă diagramă mai specifică a aplicației.

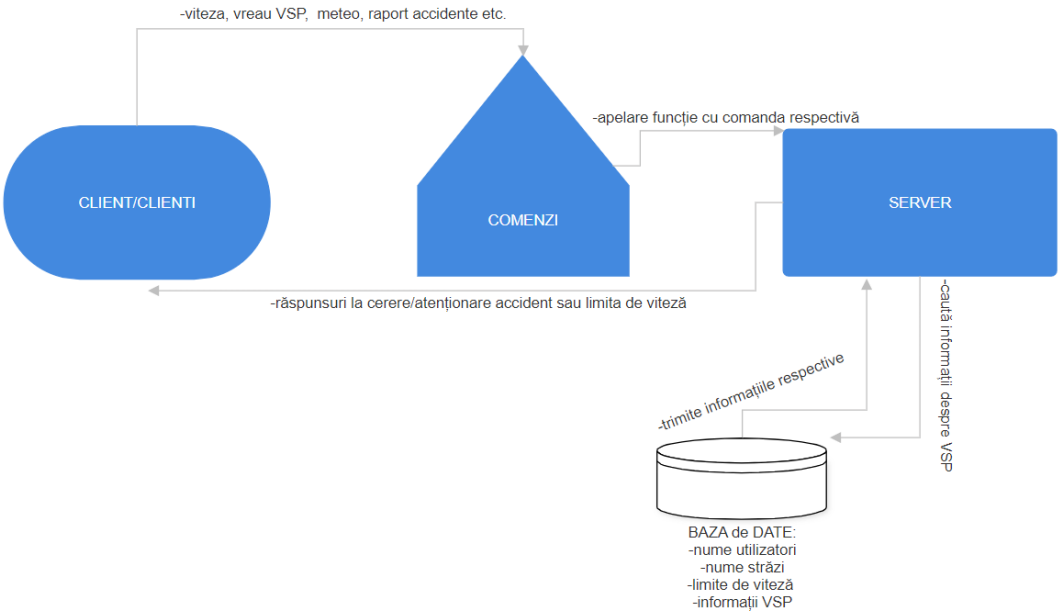

Fig. 2. Diagrama practica a aplicației

VSP este un acronim de la vreme-sport-peco. Se vede cum se trimit comenzi cu date, apoi in *"Comenzi"* se vede ce fel de comanda este(dacă nu e validă nu se recunoaste), apoi serverul prelucrează, impreună cu baza de date și se trimit informațiile aferente clienților. De retinut: *"Comenzi"* face parte din server, in diagramă am evidențiat cum se prelucrează comenziile. **!! Daca soferul nu a bifat ca vrea optiuni extra VSP, atunci cand o sa ceara detalii despre vreme/evenimente sportive/preturi la carburant de pe strada respectiva nu i se va afisa nimic.**

4 Aspecte de Implementare

- **Crearea unui socket și conectarea client-server**
  Prin socket clientul și serverul comunică . Aici clientul creează un socket.

```
int sd;
if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("[client]Eroare la socket().\n");
    return errno;
}
```

Cu `SOCK_STREAM` specificăm ca folosim TCP, iar `AF_INET` reprezintă că folosim IPv4 ca adresă. Apoi serverul se leagă de un port și ascultă cererile de conexiune.

```
int sd;
struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr(argv[1]);
```

```
server.sin_port = htons(PORT);
if (connect(sd, (struct sockaddr *) &server, sizeof(struct sockaddr)) == -1) {
    return errno;}
```

Structura `sockaddr_in` reține informații despre adresa serverului cu care clientul se conectează.

- `sin_family = AF_INET`: Folosim adrese IPv4.
- `sin_addr.s_addr = inet_addr(argv[1])`:Argv[1] este adresa dată de la tastatură. Funcția `inet_addr()` convertește adresa IP în format numeric.
- `sin_port = htons(PORT)`: Reprezintă portul unde serverul ascultă cererile.

- **Schimb de informații client - server**

Exemplu : clientul trimite un accident server-ului:

```
char mesaj[256];
sprintf(mesaj, "raport: Strada Lapusneanu: Blocaj la km 20");
write(sd, mesaj, strlen(mesaj));
```

Sau cum primeste clientul informații de la server:

```
char raspuns[256];
read(sd, raspuns, sizeof(raspuns));
printf("Raspuns de la server: %s\n", raspuns);
```

- **Thread-uri**
  O altă funcționalitate este cum gestionăm concurenta. Răpunsul este prin ThreadPoll. Fiecare cerere de la clienți va avea câte un thread pentru a se ocupa de ele.

  ```
  void threadCreate(int i) {
      pthread_create(&threadsPool[i].idThread, NULL, &treat, (void *)i);
  }
  ```

  Se creează thread-urile cu `pthread_create()`(biblioteca `pthread.h`). **ThreadPool** este un array de structuri de tip `Thread`, iar fiecare structură conține informații despre: **idThread** – ID-ul thread-ului, **thCount** – numărul de conexiuni ale thread-ului.
  Apoi am salvat în **ThreadPool** ID-ul acestuia. Al doilea argument `NULL` din apelul `pthread_create` indică faptul că nu am specificat funcționalități suplimentare pentru thread. Funcția `treat` este apelată de fiecare thread. Ea primește ca argument `i` (convertit la `void*` deoarece funcția `pthread_create` vrea acest lucru).
  Voi folosi **mutex** pe `accept()`. Blocăm accesul prin mutex pentru a nu fi mai multe fire de execuție care gestionează aceeași conexiune în același timp.

  ```
  pthread_mutex_lock(&mlock);
  client = accept(sd, (struct sockaddr *)&from, &length);
  pthread_mutex_unlock(&mlock);
  ```

  În server, am implementat un thread separat care verifică și trimite clienților activi, la o frecvență de 1 minut, viteza cu care se deplasează pe strada respectivă. De asemenea, notifică dacă aceștia se află în limita legală sau nu. Acest thread rulează în paralel cu celelalte operațiuni ale serverului, utilizând funcționalitățile oferite de biblioteca `pthread.h`.
  Codul pentru crearea acestui thread este următorul:

```
pthread_t speed_thread;
if (pthread_create(&speed_thread, NULL, frecventa_viteza, NULL) != 0) {
    perror("Eroare la crearea thread-ului pentru viteza");
    exit(EXIT_FAILURE);
}
```

Funcția `pthread_create` creează un nou thread care execută funcția `frecventa_viteza`, responsabilă de:

- Citirea din baza de date a clienților activi, inclusiv viteza și strada curentă.
- Compararea vitezei fiecărui client cu limita legală specificată pentru strada respectivă.
- Trimiterea notificărilor către clienții activi prin socket, informându-i dacă au depășit limita sau dacă sunt în regulă.

Thread-ul verifică aceste condiții la fiecare minut, utilizând o pauză `sleep(60)` între iterații. Acest design permite rularea automată și periodică a verificărilor, fără a afecta alte operațiuni ale serverului.

- **MySQL**
  Serverul salvează și preia informații din baza noastră de date. Conexiunea se face cu ajutorul bibliotecii `mysql/mysql.h`, iar interacțiunea cu baza de date se realizează prin interogări SQL. Se extrag date precum limita de viteză, condițiile meteo, prețurile pentru combustibil etc.

  Am implementat o funcție pentru inițializarea bazei de date:
  ```
  MYSQL *init_mysql_connection() {}
  ```

  Crearea bazei de date și a uneia dintre tabele cu inserări:
  ```
  CREATE DATABASE baza_date;

  USE baza_date;

  CREATE TABLE Soferi (
      id INT AUTO_INCREMENT PRIMARY KEY,
      nume VARCHAR(50) NOT NULL,
      strada VARCHAR(50),
      viteza INT DEFAULT 0,
      activ BOOLEAN DEFAULT 0,
      socket_descriptor INT
  );
  ```

Dupa cum se vede, baza de date se cheama baza_date.

- **Funcția comenzi:**
  În această funcție implementez toate comenzile pe care un șofer le poate scrie: bifare opțiuni VSP, debifare opțiuni VSP, schimb pe strada: -, update viteză -, raportează accident: -, etc.
  O parte din comenzi:
  ```
  void comenzi(int client, MYSQL *conn, const char *command, const char *name) {
      // Creare conexiune cu baza de date și prelucrare

      if (strncmp(command, "schimb pe strada:", 17) == 0) {
          // Aici, când șoferul schimbă strada, i se comunică și limita de viteză.
          // Șoferul poate ulterior să dea update viteza pentru a schimba viteza curentă.
      }
      else if (strncmp(command, "bifez optiuni extra:", 20) == 0) {
  ```

```
            // Verific ce opțiuni VSP dorește șoferul și modific în tabela "Optiuni"
            // valoarea 1 pe linia corespunzătoare clientului respectiv.
        }
        else if (strncmp(command, "raporteaza accident:", 20) == 0) {
            // Se preia strada pe care șoferul a raportat accidentul, verificăm
            // dacă strada este validă în baza de date. Dacă este, schimbăm
            // limita de viteză la 30 km/h și trimitem notificarea tuturor
            // șoferilor activi.
        }
        else if (strncmp(command, "update viteza", 13) == 0) {
            // Șoferul poate schimba viteza curentă. Dacă depășește limita legală,
            // este avertizat.
        }
        else if (strncmp(command, "debifare optiuni", 16) == 0) {
            // Dacă șoferul nu mai dorește opțiuni VSP, acestea sunt debifate.
        }
        else if{  ...

        }
    }
```

- **Funcție golire:**
  Aceasta este o funcție foarte importantă. Când un șofer vrea să iasă și să intre din nou, opțiunile active din tabelele `Soferi` și `Optiuni` (VSP) pot rămâne valide, adică setate pe `1`. Această funcție se asigură că toate bifările extra vor fi resetate la `0` la deconectare, la fel și coloana `activ`.

  Codul funcției:

```
    void golire(int signum) {
        printf("\nCurățare resurse...\n");

        // Resetare starea de activitate a șoferilor
        char query[256];
        snprintf(query, sizeof(query), "UPDATE Soferi SET activ = 0");
        if (mysql_query(global_conn, query)) {
            perror("Eroare la actualizarea stării deconectate în baza de date.\n");
        }

        // Resetare opțiuni extra (VSP) pentru toți șoferii
        snprintf(query, sizeof(query), "UPDATE Optiuni SET vreme = 0, evenimente_sportive = 0,
        if (mysql_query(global_conn, query)) {
            perror("Eroare la resetarea opțiunilor în baza de date.\n");
        }
    }
```

- **Scenarii Reale de Utilizare**
- **Schimbarea limitei de viteză**

Pe strada Lăpușneanu s-a făcut aglomeratie. Limita de viteză se schimbă. Clientul va obține informațiile actualizate de la server, care va verifica in baza de date pentru a confirma schimbarea limitei de viteză.

- **Condiții meteo**
  Șoferii care călătoresc pe distanțe mari pot verifica condițiile meteo de pe traseul lor. Vor să știe daca este soare, dacă plouă, dacă ninge. Serverul va furniza informații despre vreme, extrase din baza de date.
- **Prețuri la stațiile PECO**
  Când un șofer ajunge într-un oraș și are nevoie de combustibil, se poate conecta la server pentru a obține informații despre prețurile de la stațiile PECO.
- **Raportarea unui accident**
  Un șofer poate raporta un accident rutier la server, care va salva informațiile în baza de date și le va transmite către toți ceilalți clienți conectați pentru a-i avertiza despre blocajul din trafic.

Afisarea vitezei la o frecventa de 1 minut
La fiecare minut i se va afisa fiecarui client activ viteza si daca este peste limita de viteza de pe aceea strada sau nu.

5 Concluzie

Proiectul *"Monitorizarea Traficului"* este o aplicație complexă, care testează abilitatea de a implementa un server concurent și de a interacționa cu o bază de date MySQL. Proiectul poate fi considerat un frate mai mic al aplicațiilor Google Maps sau Waze, testând cunoștințele legate de comunicarea server-client, protocoale și baze de date.
Ca **posibile imbunătățiri:**

- Permite utilizatorului să aleagă traseul dorit și să obțină informații despre consumul de combustibil în funcție de condițiile meteo (Ex: pe timp de ploaie masina va consuma mai mult).
- Implementarea unui sistem de punctaj pentru raportarea accidentelor rutiere. Șoferii vor primi puncte daca sunt activi pe aplicație și dau informații corecte. Dacă dau informații greșite li se va scădea din punctaj.

References

1. Lenuta Alboaie, Rețele de Calculatoare, Anul 2024. Disponibil la: `https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php`.
2. Stiri din Lume. Disponibil la: `https://www.stiridinlume.ro/tehnologie-stiinta/cine-a-creat-waze-cum-a-luat-nastere-cea-mai-populara-aplicatie-de-navigatie-8324.html`
3. Geek for Geeks. Disponibil la: `https://www.geeksforgeeks.org/mysql-tutorial/`
4. SmartDraw. Disponibil la: `https://www.smartdraw.com/flowchart/flowchart-maker.htm?id=376435&gad_source=1&gclid=CjwKCAiA9bq6BhAKEiwAH6bqoLfDb5JXJMSGfyEyPU4SQuF9_3yOchR-glajjsIX6dEmFwMpNOuJJxoCc2oQAvD_BwE`
5. Overleaf. Disponibil la: `https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes`
6. Medium. Disponibil la: `https://medium.com/@nchumbadze/threadpool-in-practice-7d07f169d9a8`
7. Globema. Disponibil la: `https://www.globema.ro/scurt-ghid-pentru-intelegerea-produselor-din-platforma-googl`
8. MySQL : `https://dev.mysql.com/doc/mysql-tutorial-excerpt/8.0/en/`