Traffic management

Spiridon Roberto-Riccardo

Universitatea Alexandru Ioan Cuza, Facultatea de Informatica
robertosissues1@gmail.com

1 Introduction

The "Traffic Monitoring" project is a complex project that manages road traffic and provides useful information to drivers. The purpose of this server-client application is to significantly improve road safety. Functionalities include reporting incidents, receiving information from drivers about weather conditions, sporting events, fuel prices at PECO stations (diesel and gasoline), unchecking these options if the driver wants, but also the recommended speed on a certain section of the road**. Another functionality is checking the speed at a frequency of 1 minute.**

Of those listed above, these functionalities may remind you of the Google Maps application or, more recently, Waze. This was bought by Google 7 years after its launch. The application focuses on a concurrent TCP server and its interaction with multiple clients. Find out in the following chapters why we chose a concurrent TCP.

- ### Applied Technologies

As I said above, we will use a concurrent TCP as well as a database.

• **TCP:** I will explain why I chose TCP over TCP or UDP. Let's imagine that drivers and the server are people who send each other letters. They contain information. Let's say that a driver sends a letter to the server to report an accident. TCP guarantees that the letter, along with the information in it, reaches the recipient without loss. Even in the worst case, if the letter is lost, it will resend it to the recipient until it arrives safely. On the other hand, the courier (via UDP) no longer asks if you received the letter, he sends very quickly and does not take into account the integrity of the information in the letters. We could say that TCP is a careful courier and UDP a hasty courier.

• **MySQL:** The database used in this project is MySQL because it is powerful and manages a large number of data. It is widely used in industry, using the SQL (Structure Query Language) language. Interaction with the database will be done through SQL queries, using the mysql/mysql.h library. In this database I will retain information about speeds, customer names (with initial information about the speed at which they travel and on which street), sports events, accident reports but also about weather conditions.

2 Aplication Structure

As I said in the "Introduction", the application consists of two main entities: the server and the clients. The server uses a concurrent TCP protocol, so that more than one client can communicate with the server simultaneously. I will first start with the diagram to give you an idea of how data is sent and read, but also of the interaction with a database.

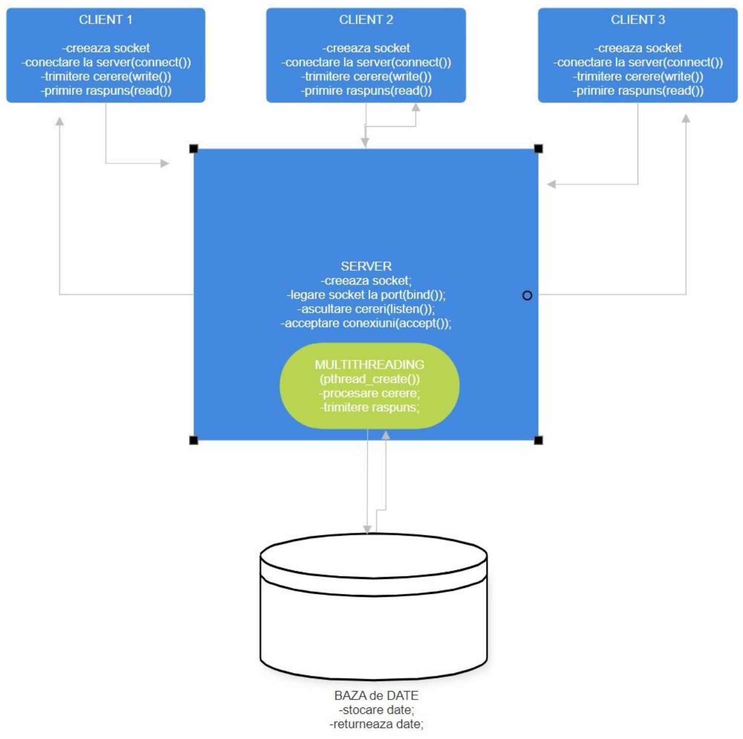

Fig.1. Architectural diagram of the app

- **General diagram of the app**

The server creates a socket using the **socket()** function and binds it to a port using bind(). Then, the server listens for connections using listen() and accepts connections using accept().

This is where **Multithreading** comes in. What does this mean? Each request from a client is taken by one of the threads to be processed in parallel. So for each accepted connection, a thread is used. I have implemented a thread in the server to check the speed of each driver at a frequency of 1 minute and it is displayed to each client on the screen. It will show them whether they have exceeded the speed limit on the street they are on or not. This thread is created in the client.

Now our database appears where valuable information for drivers is stored: about the weather, speed limit, sporting events, username (with street name and initial speed) but also the price for fuel. Then the server after taking this information sends answers to the clients. I will initialize the MySQL database in the server and I will have a separate file in which I create the database with tables and inserts into them. (Table - Drivers, Options, Streets).

In turn, the client creates a socket to connect to the server using the connect() function. Once connected, the client can send information requests and report crashes. The server receives these requests, processes them, and sends the corresponding responses to the clients. In the client, we have implemented a thread that displays each response from the server, so that there are no overlapping problems.

Now, after I have explained the theoretical part of the concurrent **TCP functionality** in our application, I will present another more specific diagram of the application.

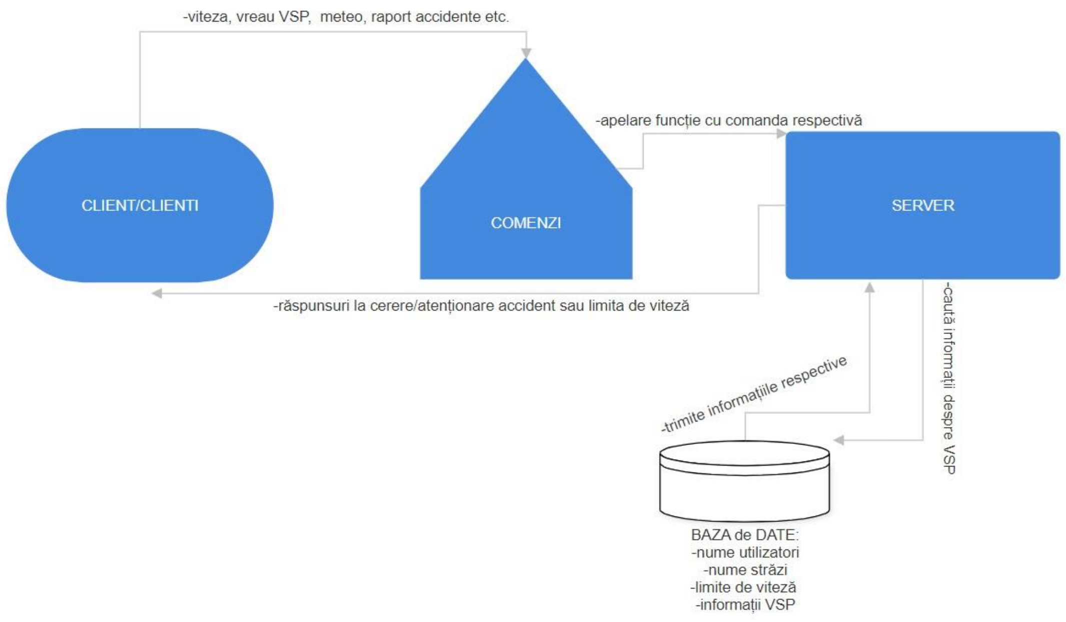
-viteza, vreau VSP, meteo, raport accidente etc.
CLIENT/CLIENTI
COMENZI
-apelare funcție cu comanda respectivă
SERVER
-răspunsuri la cerere/atenționare accident sau limita de viteză
-trimite informațiile respective
-caută informații despre VSP
BAZA de DATE:
-nume utilizatori
-nume străzi
-limite de viteză
-informații VSP

Fig.2. Practical diagram of the app

VSP is an acronym for weather-sport-peco. You can see how orders are sent with data, then in "Orders" you can see what kind of order it is (if it is not valid it is not recognized), then the server processes it, together with the database, and sends the related information to the customers. Remember: "Orders" is part of the server, in the diagram I have highlighted how the orders are processed**. !! If the driver has not checked that he wants extra VSP options, when he asks for details about weather/sports events/fuel prices on the respective street, nothing will be displayed.**

3 Aspects of implementation

• Creating a socket and connecting client-server
The client and server communicate via socket. Here the client creates a socket.

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

With SOCK STREAM we specify that we are using TCP, and AF INET means that we are using IPv4 as the address. Then the server binds to a port and listens for connection requests.

```
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;}
```

The sockaddr structure holds information about the address of the server to which the client connects.

sin family = AF INET: Folosim adrese IPv4.

sin addr.s addr = inet addr(argv[1]):Argv[1] is the address given from the keyboard. The inet addr() function converts the IP address to numeric format.

sin port = htons(PORT): Represents the port where the server listens for requests.

- **Client - server information exchange**

Example: client sends a crash to the server:

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

Or how the client receives information from the server:

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

- **Threads**

Another feature is how we handle concurrency. The answer is through ThreadPoll.
Each client request will have a thread to handle it.

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

Threads are created with pthread create() (pthread.h library). ThreadPool is an array of Thread structures, and each structure contains information about: idThread – the thread ID, thCount – the number of thread connections.
Then we saved its ID in ThreadPool. The second NULL argument in the pthread create call indicates that we did not specify additional functionality for the thread. The treat function is called by each thread. It receives i as an argument (converted to void* because the pthread create function wants this).
I will use mutex on accept(). We block access through mutex so that there are not more than one thread managing the same connection at the same time.

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

In the server, we implemented a separate thread that checks and sends to active clients, at a frequency of 1 minute, the speed at which they are moving on the respective street. It also notifies them whether they are within the legal limit or not. This thread runs in parallel with the other server operations, using the functionalities provided by the pthread.h library.

The code for creating this thread is the following:

```
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);
}
```

The pthread create function creates a new thread that executes the frequent speed function, which is responsible for:

- Reading the active clients from the database, including their speed and current street.

Comparing each client's speed with the legal limit specified for that street.

- Sending notifications to active clients via sockets, informing them whether they have exceeded the limit or whether they are OK.

The thread checks these conditions every minute, using a sleep(60) pause between iterations. This design allows the checks to run automatically and periodically, without affecting other server operations.

- **MySQL**

The server saves and retrieves information from our database. The connection is made using the mysql/mysql.h library, and the interaction with the database is done through SQL queries.
Data such as speed limits, weather conditions, fuel prices, etc. are extracted.

I implemented a function to initialize the database:

```
MYSQL *init_mysql_connection() {}
```

Creating the database and one of the tables with inserts:

```
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
);
```

As you can see, the database is called baza_date.

- **Command function:**

In this function I implement all the commands that a driver can write: check VSP options, uncheck VSP options, change on the street: -, update speed -, report accident: -, etc.

Some of the commands:

```
void comenzi(int client, MYSQL *conn, const char *command, const char *name) {
    // Create conexion with database

    if (strncmp(command, "schimb pe strada:", 17) == 0) {
    // Here, when the driver changes the street, he is also informed of the speed limit.
    // The driver can later update the speed to change the current speed.}
```

```
        else if (strncmp(command, "bifez optiuni extra:", 20) == 0) {
            // Check which VSP options the driver wants and modify them in the "Options" table // the
            value 1 on the line corresponding to the respective customer.
        }
        else if (strncmp(command, "raporteaza accident:", 20) == 0) {
        // We take the street where the driver reported the accident, check
        // if the street is valid ^in the database. If it is, we change
        // the speed limit to 30 km/h and send the notification to all // active drivers.
        }
        else if (strncmp(command, "update viteza", 13) == 0) {
        // The driver can change the current speed. If he exceeds the legal limit, // he is warned.
        }
        else if (strncmp(command, "debifare optiuni", 16) == 0) {
        // If the driver no longer wants VSP options, they are unchecked.

        }else if{ ...

        }
    }
```

• **Clear Function:**

This is a very important function. When a driver wants to log out and log in again, the active options in the Drivers and Options (VSP) tables can remain valid, i.e. set to 1. This function ensures that all extra checkmarks will be reset to 0 upon logout, as well as the active column.

Function code:

```
void golire(int signum) { printf("\nCurˇat¸are resurse...\n");
        // Reset driver activity status char query[256];
        snprintf(query, sizeof(query), "UPDATE Soferi SET activ = 0"); if (mysql_query(global_conn,
        query)) { perror("Eroare la actualizarea stˇarii deconectate ^ın baza de date.\n");
        }

        // Reset extra options(VSP) for all drivers
        snprintf(query, sizeof(query), "UPDATE Optiuni SET vreme = 0, evenimente_sportive = 0, if
        (mysql_query(global_conn, query)) { perror("Eroare la resetarea opt¸iunilor ^ın baza de date.\n");
        }
    }
```

- **Real-World Usage Scenarios**

  •Speed limit change:

  There is a traffic jam on Lăpușneanu Street. The speed limit is changing. The client will get the updated information from the server, which will check the database to confirm the speed limit change.

  •Weather conditions:

  Drivers traveling long distances can check the weather conditions along their route. They want to know if it is sunny, if it is raining, if it is snowing. The server will provide weather information, extracted from the database.

  • Prices at PECO stations:

When a driver arrives in a city and needs fuel, he can connect to the server to obtain information about prices at PECO stations.

- **Reporting an accident:**
  A driver can report a traffic accident to the server, which will save the information in the database and transmit it to all other connected clients to warn them about the traffic jam.

Speed display at a frequency of 1 minute : Every minute, each active client will be shown their speed and whether they are over the speed limit on that street or not.

4 Conclusion

The "Traffic Monitoring" project is a complex application that tests the ability to implement a concurrent server and interact with a MySQL database. The project can be considered a little brother to Google Maps or Waze applications, testing knowledge related to server-client communication, protocols, and databases.

Possible improvements:

• Allow the user to choose the desired route and obtain information about fuel consumption depending on weather conditions (Ex: during rain the car will consume more).

• Implement a scoring system for reporting road accidents. Volunteers will receive points if they are active on the application and provide correct information. If they provide incorrect information, their score will be deducted.

References

1. Lenuta Alboaie, Reț ele 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=CjwKCAiA9bq6BhAKEiwAH6bqoLfDb5JXJMSG fyEyPU4SQuF9_ 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/