Raport Tehnic

Pricopi Nicoleta Carmen Bianca

Facultatea de Informatica, Universitatea "Alexandru Ioan Cuza", Iasi
biancapricopi1@gmail.com
https://www.info.uaic.ro/

Introducere

Obiectivul prezentului raport este de a oferi o descriere detaliata in vederea realizarii unui sistem de monitorizare a traficului simulat. Include atat informatii tehnice, legate de implementare, cat si detalii functionale. Monitorizarea traficului este un software creat cu scopul de a gestiona si facilita experienta soferilor. Pentru a accesa serviciile oferite de software, un sofer trebuie sa isi creeze un cont de utilizator sau sa se conecteze la un cont deja existent. Odata conectata la sistem fiecare masina va raporta viteza curenta de deplasare cu o frecventa de 1 minut, si va primi feedback legat de limitele de viteza de pe portiunea de drum pe care circula, dar si de eventualele blocaje din trafic. In cazul producerii unui accident, orice utilizator conectat poate raporta accidentul, dupa care sistemul va trimite un mesaj de avertizare catre toti soferii conectati.
In cazul sesizarii unui ambuteiaj, orice utilizator conectat poate raporta ambuteiajul, iar sistemul va tine de cont de acesta in momentul verificarii limitei de viteza corespunzatoare locatiei respective.
Sistemul ofera posibilitatea afisarii traseului minim de urmat pentru a ajunge dintr-o locatie in alta.
De asemenea, sistemul poate furniza informatii legate de conditiile meteorologice, evenimente sportive sau preturile combustibilului la diferite statii de alimentare.

Proiectul nu prezinta interfata grafica, iar interactiunea dintre soferi si sistem are loc la nivel de consola. Soferii reprezinta clientii, iar sistemul reprezinta server-ul. Server-ul este concurent, insemnand ca poate servi mai multi clienti in acelasi timp. Pentru a asigura concurenta server-ului se folosesc thread-uri, server-ul fiind multithreaded. In ceea ce priveste informatiile oferite de sistem, pentru stocarea acestora se foloseste o baza de date relationala, anume MySQL.

Tehnologiile utilizate

Comunicarea dintre server si clienti se realizeaza folosind TCP (Transmission Control Protocol). De ce TCP? TCP este un protocol orientat conexiune, deci odata stabilita conexiunea, informatia poate fi transmisa in doua directii. TCP verifica posibilele erori si asigura transmiterea completa si in ordine a informatiei. Foarte important, pachetele trebuie trimise in ordine. Dupa ce s-a conectat la sistem, un sofer poate solicita oricate comenzi doreste, pana cand se

inchide conexiunea.

Pentru concurenta server-ului se utilizeaza thread-uri (fire de executie) datorita performantei. Pentru fiecare client care se conecteaza la server se creeaza un nou thread, care va servi clientul respectiv. Functia asociata thread-urilor se va numi handleClient(). Un proces poate avea mai multe thread-uri, care au shared memory cu procesul care le-a creat. Din aceasta cauza pot aparea race conditions daca doua sau mai multe thread-uri acceseaza si modifica aceeasi zona de memorie (de exemplu aceeasi variabila globala) in acelasi timp. Aceasta situatie este intalnita in cazul comenzii **get-directions** deoarece algoritmul lui djikstra se foloseste de doua structuri de date globale **indexForPath** si **path**. Pentru a evita race conditions se foloseste **pthread_mutex_t**.
Cum fiecare thread are propriul stack nu apar race conditions in ceea ce priveste variabilele declarate in interiorul functiei **handleClient()**.
"In programarea C sub Linux se utilizeaza interfata pthreads definita conform standardelor POSIX (la compilarea surselor cu gcc se va adauga optiunea -l pthread pentru a linka libraria de threaduri)." Biblioteca utilizata este pthread.h.

Pentru a asigura primirea permanenta a pachetelor de la server, clientul foloseste, de asemenea thread-uri. In acest caz nu apar race conditions. Functia asociata thread-urilor este **recvServerReply()**.

Pentru stocarea datelor se utilizeaza baza de date relationala MySQL. Baza de date va contine informatii despre fiecare utilizator (ID, username, locatia), o parte dintre strazile din Iasi, categorii de drumuri (nationale, europene, autostrazi) precum si viteza maxima corespunzatoare, informatii legate de conditiile meteorologice, evenimente sportive si preturile combustibilului din diferite statii de alimentare, dar si accidentele si ambuteiajele raportate. Baza de date se va numi **TrafficManagement**. Biblioteca utilizata este **mysql.h**.

Arhitectura Aplicatiei

Clientul isi creeaza un socket care are parametrii PF_INET, SOCK_STREAM si IPPROTO_TCP, dupa care initializeaza conectarea socketului sau la un socket al serverului prin apelarea functiei connect(). Odata realizata conexiunea clientul transmite comenzi prin intermediul functiei send() si citeste raspunsul prin recv(). Pentru a inchide conexiunea se apeleaza close().
Serverul isi creeaza un socket care are aceiasi parametri, dupa care il asociaza cu o adresa si un port prin bind(). Dupa bind(), serverul asteapta conexiuni, apeleaza functia listen(). Cand un client este conectat, accept(), serverul creeaza alt socket pentru a comunica cu acesta. Comunicarea se realizeaza prin send() si recv(), iar terminarea conexiunii se realizeaza prin apelul functiei close().
Functionalitatea clientului

Clientul citeste o comanda de la tastatura, creeaza un pachet care contine lungimea comenzii si comanda, separate prin "#", pe care il trimite serverului, dupa care citeste pachetul primit de la server, il decodifica, dupa care afiseaza raspunsul in consola. Repeta procedeul pana cand utilizatorul inchide

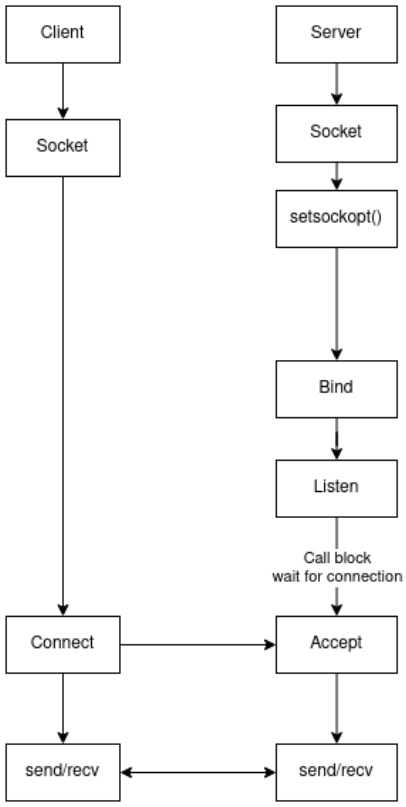
Client
Server
Socket
Socket
setsockopt()
Bind
Listen
Call block
wait for connection
Connect
Accept
send/recv
send/recv

Fig. 1. TCP/IP

brusc aplicatia sau foloseste comanda **quit**.

Functionalitatea server-ului

Server-ul are cate un thread pentru fiecare client, astfel poate servi mai multi clienti in acelasi timp. Citeste pachetul primit de la client, il decodifica, verifica ce comanda este, dupa care construieste un pachet care contine o serie de flaguri: quit (0 sau 1), loggedIn (0 sau 1), lastCommandIsSpeed (0 sau 1), accident (0 sau 1), validStreet (0 sau 1), la care se adauga lungimea mesajului si mesajul efectiv, toate fiind separate prin "#". Trimite pachetul catre client. In cazul in care clientul trimite o comanda gresita, serverul ii trimite un mesaj de eroare si ii solicita sa introduca o noua comanda. Verificarea dimensiunii corecte a mesajului se realizeaza in server. De exemplu username-ul este un sir de caractere cu o dimensiune maxima de 50 de caractere. Daca un client isi intrerupe brusc conexiunea, ceilati clienti nu sunt afectati, iar server-ul ramane in continuare deschis. Server-ul poate accepta 20 de clienti concomitent, iar dupa 100 de conectari efectuate trebuie sa fie restartat. Se va afisa un mesaj de avertizare in momentul in care este atinsa limita de 100, iar clientii care incearca sa se conecteze vor primi un mesaj de eroare referitor la faptul ca server-ul nu este disponibil.

Protocolul cuprinde urmatoarele comenzi:
Toate comenzile sunt executate de server, clientul doar le citeste.
login : username (verificarea existentei username-ului in baza de date se face face de catre server, in cazul in care username-ul exista deja utilizatorul este conectat cu succes, iar in caz contrar i se va solicita sa isi faca un cont)
register : username (in cazul in care username-ul exista deja, utilizatorului i se va trimite un mesaj de avertizare, daca nu, i se va trimite un mesaj de confirmare)
logout (utilizatorul va fi deconectat)
quit (va deconecta clientul de la server)
help (afiseaza sintaxa si descrierea comenzilor disponibile)

Urmatoarele comenzi vor putea fi executate doar daca clientul este conectat la sistem.
delete-account (contul utilizatorului conectat va fi sters)
speed : < **viteza** > (fiecare client va fi instiintat de client ca trebuie sa trimita viteza cu care circula cu o frecventa de un minut. Server-ul va verifica daca viteza este regulamentara)
accident : < **street** > (fiecare client poate raporta un accident pe care serverul il va inregistra in baza de date si va trimite un mesaj catre toti utilizatorii conectati la sistem)
no-accident : < **street** > (fiecare client care observa ca pe o anumita strada accidentul a fost solutionat, poate sterge accidentul din sistem)
all-accidents (pentru a vedea toate accidentele raportate)
traffic-jam : < **street** > (pentru a raporta un ambuteiaj)
no-traffic-jam : < **street** > (pentru a sterge un ambuteiaj)
all-traffic-jams (pentru a vedea toate ambuteiajele raportate)
newsletter : < **weather** >, < **sport** >, < **fuel** > (clientul poate solicita abonarea la una sau mai multe categorii, iar serverul ii va trimite informatii despre op-

tiunea aleasa)
get-directions : $<$ **startStreet** $>\rightarrow<$ **destinationStreet** $>$ (pentru a vedea ruta dintre startStreet si destinationStreet)

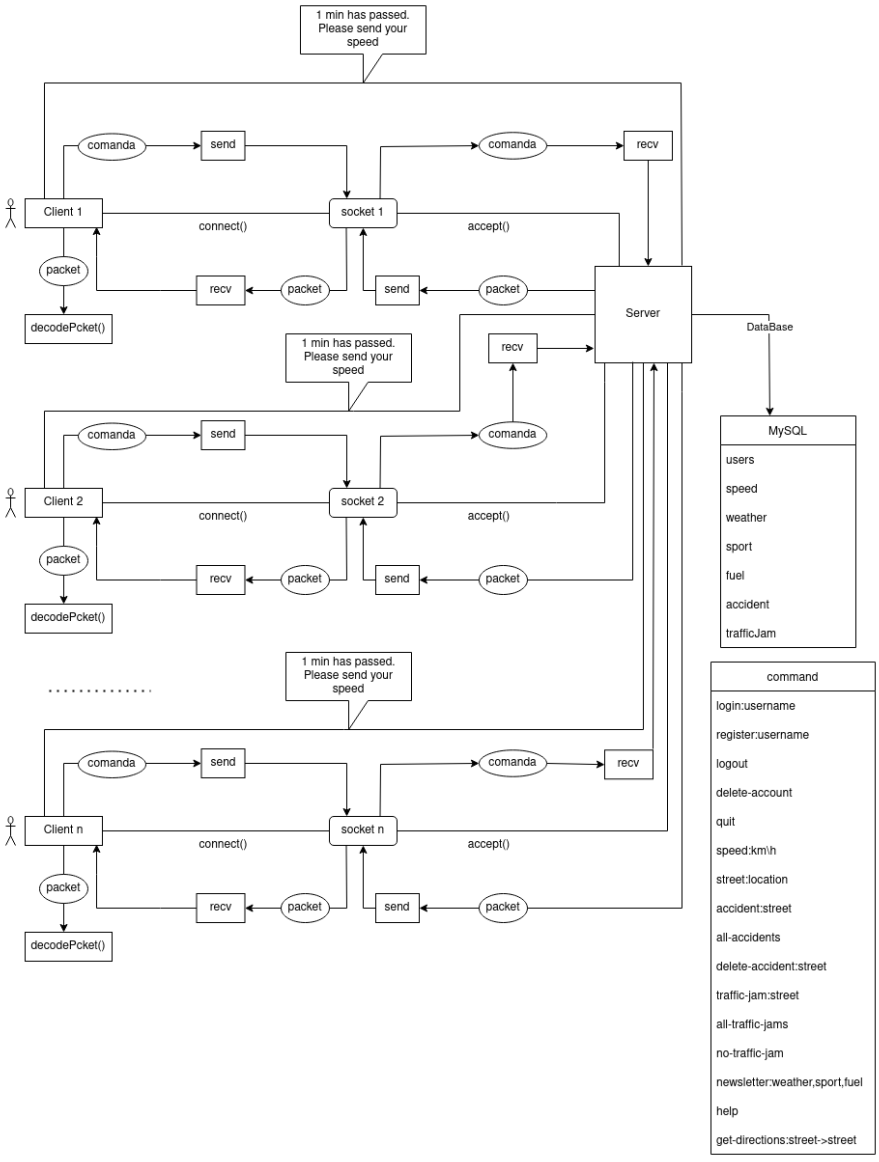

Fig. 2. Diagrama proiectului

Detalii de implementare

Functia makePacket() creeaza un pachet pe care server-ul urmeaza sa i-l trimita clientului. Asemanator, functia makePacket() este utilizata si in client pentru a face pachetul ce trebuie trimis server-ului, dar va avea ca parametri doar lungimea comenzii, comanda efectiva si bufferul packet.

```
void makePacket(char commandMessage[], int quit, int loggedIn
    ,int lastCommandIsSpeed, int ifAccident
    ,int ifValidStreet, int length, char packet[])
{
        char intToChar[10];
        sprintf(intToChar, "%d", quit);
        strcpy(packet, intToChar);
        strcat(packet, "#");
        bzero(intToChar, sizeof(intToChar));
        sprintf(intToChar, "%d", loggedIn);
        strcat(packet, intToChar);
        strcat(packet, "#");
        bzero(intToChar, sizeof(intToChar));
        sprintf(intToChar, "%d", lastCommandIsSpeed);
        strcat(packet, intToChar);
        strcat(packet, "#");
        bzero(intToChar, sizeof(intToChar));
        sprintf(intToChar, "%d", ifAccident);
        strcat(packet, intToChar);
        strcat(packet, "#");
        bzero(intToChar, sizeof(intToChar));
        sprintf(intToChar, "%d", ifValidStreet);
        strcat(packet, intToChar);
        strcat(packet, "#");
        bzero(intToChar, sizeof(intToChar));
        sprintf(intToChar, "%d", length);
        strcat(packet, intToChar);
        strcat(packet, "#");
        strcat(packet, commandMessage);
        strcat(packet, "#");
}
```

Pachetul respecta urmatoarea ordine, prima parte va contine flag-urile quit (0 sau 1), astfel clientul isi va intrerupe sau nu executia, loggedIn (0 sau 1), clientul va porni sau nu alarma pentru a-i cere clientului viteza cu o frecventa de un minut, lastCommandIsSpeed (0 sau 1), clientul va reseta alarma, accident (0 sau 1), pentru a face cate recv-uri este nevoie, validStreet (0 sau 1), pentru a putea cere strada pana cand utilizatorul introduce o strada valida, iar a doua parte va contine dimensiunea mesajului in bytes, iar a treia parte va contine

mesajul efectiv.
Toate partile sunt separate de un "#".

Functia decodePacket() decodifica pachetul primit de la server. Prin numararea separatorilor se identifica fiecare parte componenta a pachetului.
Aceasta este folosita in mod similar atat in client.c, cat si server.c, deoarece comunicarea dintre client si server este codificata folosind logica pachetelor.

```
void decodePacket(char commandPacket[], char command[], int *commandLength)
{
  int i = 0, j = 0;
  int packetLength = strlen(commandPacket);
  char tempNum[10];
  bzero(tempNum, sizeof(tempNum));
  int countSharp = 0;
  while(i < packetLength && countSharp < 1)
  {
    if(commandPacket[i] == '#')
    {
      countSharp++;
      if(countSharp == 1)
      {
        *commandLength = atoi(tempNum);
      }
      bzero(tempNum, sizeof(tempNum));
      j = 0;
      i++;
    }
    tempNum[j++] = commandPacket[i++];
  }
  i--;
  j = 0;
  while(commandPacket[i] != '#' && i < packetLength)
  {
    command[j++] = commandPacket[i++];
  }
}
```

Pentru comenzile care necesita doua argumente (primul argument este considerat a fi denumirea comenzii, iar al doilea cuprinde informatia necesara comenzii respective), se verifica numarul de argumente si se adauga mesajul corespunzator. Parsarea celui de-al doilea argument poate sa difere in functie de specificul fiecarei comenzi.

```
int verifyNrArguments(char comanda[])
{
        int lungCom = strlen(comanda);
        int flag = 0;
```

```
        int argumentsNum = 0;
        int i = 0;
        while(i < lungCom && comanda[i] != ':')
        {
                i++;
        }
        argumentsNum = 1;
        while(i < lungCom)
        {
                flag = 1;
                if(comanda[i] == ',')
                {
                        argumentsNum++;
                }
                i++;
        }
        if(flag)
        {
                argumentsNum++;
        }
        return argumentsNum;
}
void extractArgument(char comanda[], int index, char argument[])
{
        int lungCom = strlen(comanda);
        int i = 0, j = 0;
        if(index == 1)
        {
                while(comanda[i] != ' ' && i < lungCom)
                {
                        argument[j++] = comanda[i++];
                }
        }
        if(index == 2)
        {
                while(comanda[i] != ':' && i < lungCom)
                {
                        i++;
                }
                i++;
                while(i < lungCom)
                {
                        argument[j++] = comanda[++i];
                }
        }
```

```
        argument[j] = '\0';
}
```

In cazul in care comanda primita de server este eronata, acesta va instiinta clientul prin trimiterea unui mesaj de eroare si ii va solicita o noua comanda. Pentru ca o comanda sa fie valida aceasta trebuie sa existe in protocol si sa respecte sintaxa prestabilita.

Comenzile nu sunt case-sensitive, dar sunt whitespace-sensitive.

Baza de date este creata folosind doua fisiere C, anume createDataBase.c si tablesGestTraffic.c.

Sistemul de strazi al orasului arata astfel:

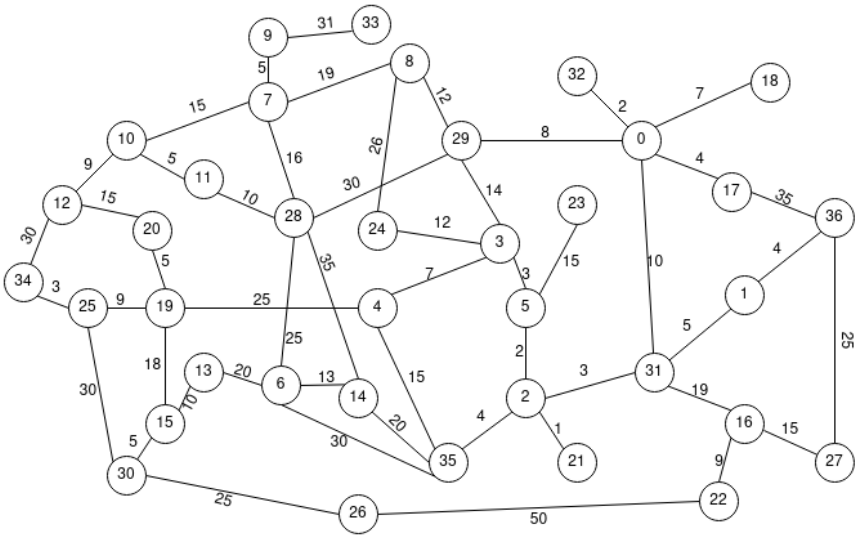

Fig. 3. Strazi

Concluzii

Monitorizarea Traficului este un proiect bazat pe implementarea unui server concurent. Realizarea concurentei este asigurata prin thread-uri. Fiecare utilizator va scrie la tastatura comenzile pe care doreste sa le primeasca, iar sistemul va interpreta fiecare input. Ramane la latitudinea soferilor sa raporteze accidente sau ambuteiaje pentru a mentine sistemul updated.

Imbunatatirile pot fi posibilitatea ca un client sa se conecteze cu o parola si adaugarea unei interfate grafice.

Bibliography

[1] Code Highlighting with minted https://www.overleaf.com/learn/latex/Code_Highlighting_with_minted
[2] Overleaf Documentation, https://www.overleaf.com/learn.
[3] Alboaie Lenuta, Panu Andrei (2021) Retele de calculatoare-Facultatea de Informatica https://profs.info.uaic.ro/ computernetworks/contact.php.
[4] Dancrumb (2011) Does local variable in thread function have separated copy according to thread? https://stackoverflow.com/questions/7387620/does-local-variable-in-thread-function-have-separe-copy-according-to-thread.
[5] POSIX Threads, http://www.csc.villanova.edu/ mdamian/threads/posixthreads.html.
[6] Matt Cook (2017) TCP vs. UDP: What's the Difference? https://www.lifesize.com/en/blog/tcp-vs-udp/
[7] Tony Li Xu, C - Implicit Declaration of Function 'inet_addr' https://www.lixu.ca/2014/09/c-implicit-declaration-of-function.html
[8] ZetCode https://zetcode.com/db/mysqlc/
[9] diagrams.net https://app.diagrams.net/
[10] Anish Ramaswamy, Keep socket open in C https://stackoverflow.com/questions/16007789/keep-socket-open-in-c
[11] CodeVault, Unix Threads in C https://www.youtube.com/c/CodeVault/playlists
[12] Tech with Tim https://youtu.be/olYdb0DdGtM
[13] Emanuel Onica https://profs.info.uaic.ro/ eonica/rc/
[14] Matei Madalin https://profs.info.uaic.ro/ matei.madalin/retele/
[15] The Ohio State Univerisity-How to Debug C Program using gdb in 6 Simple Steps https://u.osu.edu/cstutorials/2018/09/28/how-to-debug-c-program-using-gdb-in-6-simple-steps/
[16] Aditya Goel-Printing Paths in Dijkstra's Shortest Path Algorithm https://www.geeksforgeeks.org/printing-paths-dijkstras-shortest-path-algorithm/
[17] What is boolean in C? https://www.educative.io/edpresso/what-is-boolean-in-c
[18] Using colours in LaTeX https://www.overleaf.com/learn/latex/Using_colours_in$_LaTeX$