Monitorizarea traficului (A)

Simionescu Teodor

2B5

1 Introducere

Traficul este o problema reala pentru societatea actuala. Orele de varf pot bloca bucati intregi de oras, motiv pentru care s-a cautat o solutie pentru optimizare. Aceasta aplicabilite in lumea reala serveste ca motiv propriu de alegere a acestei tematici pentru proiectul final al materiei *Retele de calculatoare*. Prin urmare, acest document este **raportul tehnic** al proiectului. Conform cerintei, aceasta aplicatie gestioneaza traficul oferind soferilor informatii primite de la alti soferi.

Asemenea unui raport tehnic, mai jos se gasesc informatii referitoare la **tehnologiile folosite**, cat si **motivatia** pentru care au fost folosite.

2 Tehnologiile folosite

Pentru aceasta aplicatie protocolul **TCP** -Transmission Control Protocol- a fost folosit. Server-ul actioneaza in stil **concurent** prin utilizarea **thread-urilor**. Pentru a asigura consistenta datelor pe care le va folosi server-ul, aplicatia foloseste o baza de date **SQLite**.

Alegerea protocolului TCP in detrimentului celui **UDP** a fost facuta pe baza faptului ca se doreste o comunicare sigura intre server si client, fara lipsa de informatie. Chiar daca viteza protocolului UDP este superioara, strict pentru aceasta aplicatie, din punctul meu de vedere, are mai mult sens utilizarea protocolului TCP.

Implementarea serverului concurent fata de un server iterativ este pentru a evita situatiile in care mai multi clienti ar dori sa acceseze server-ul concomitent, ce ar fi dus la o linie de asteptare.

Prin serverul concurent, fiecare client va avea creat un thread pentru a putea fi servit in timp util. Folosirea thread-ului fata de

metoda fork este pentru a avea un control mai eficient asupra clientului. Thread-urile folosesc datele prin accesare, fata de fork ce creaza un proces complet nou in care copiaza datele pentru a le folosi ulterior. Pentru un numar suficient de mare de clienti, metoda fork trebuie optimizata foarte bine pentru a avea succesul cautat.

Baza de date a fost aleasa fata de salvarea datelor intr-un fisier oarecare pentru a asigura aplicatiei posibilitatea de a avea o mentenanta superioara a datelor.

3 Arhitectura aplicatiei

Pe scurt, acesta este raportul tehnic al aplicatiei *Monitorizarea traficului* ce foloseste protocolul TCP intre client si server, cu o configuratie implementata a serverului de a servi concurent, prin thread-uri.

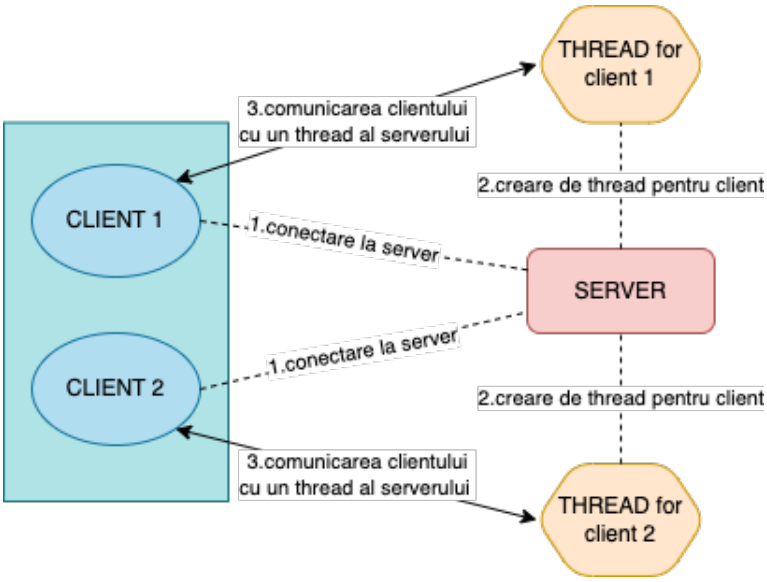


Fig. 1. Functionarea server-ului in mod concurent prin thread-uri

TCP foloseste **socketuri** pentru comunicarea intre client si server. In client, socket-ul a fost implementat folosind tehnologia fork. Comunicarea dintre client spre server este implementata in procesul

tata, iar raspunsul dintre server spre client este controlat in procesul copil.

Baza de date poate fi interogata de server. Pentru a modifica date, clientul trimite informatiile respective spre server iar acesta se va ocupa pentru a implementa schimbarile.

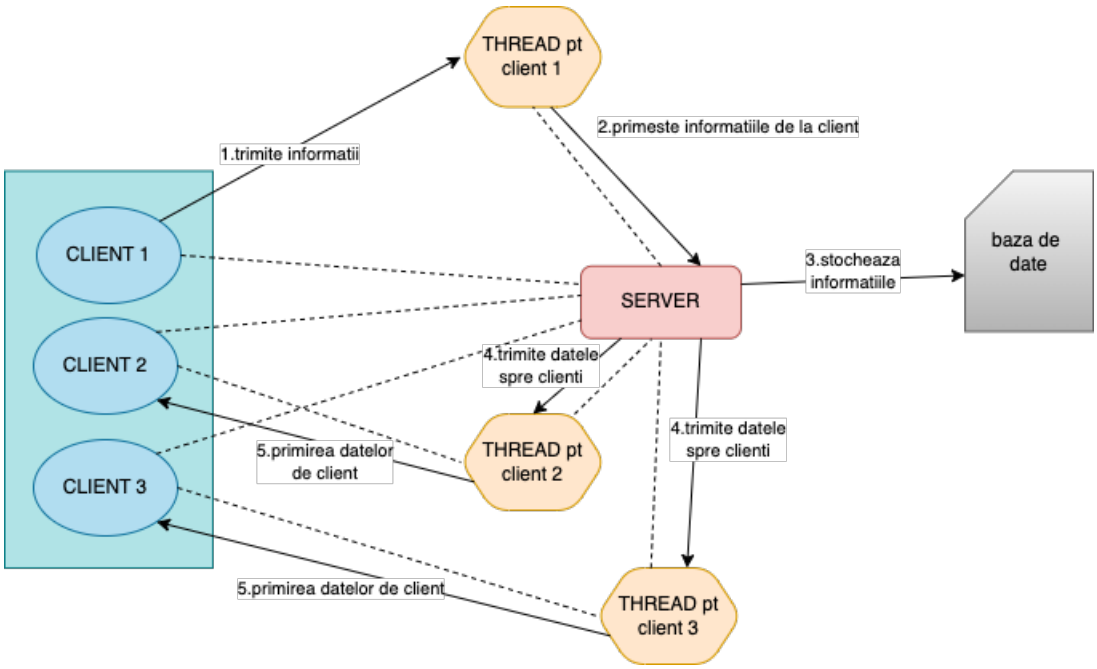


Fig. 2. Parcurgerea datelor de la client spre server, transmiterea datelor de la server spre restul clientilor

4 Detalii de implementare

4.1 Comunicarea client-server

Aceasta aplicatie confera 2 optiuni unui client: *Interogare* si *Raportare*.

Interogare creaza o comunicare intre client si server de tipul citire, unde server-ul trimite informatii catre client.

Optiunea *Interogare* ofera urmatoarele suboptiuni:

– *Viteza cu care un client trebuie sa mearga*
Aceasta informatie este trimisa catre client la un interval de un minut.

```c
/* the client can either ask for details of a new street or sent receive the details of his actual street,
if there is nothing that is introduced in the stdin for 15 seconds, the program will sent the actual street*/
int ready;
if ((ready = select(1, &readfds, NULL, NULL, &tv)) < 0)
{
   perror("[Client] Select error()\n");
   return -1;
}
else if (ready)
  read(0, buf, sizeof(buf)); /*take the new command*/
else
   strcpy(buf, "Change: S:Soseaua Toma_Cozma\n"); /*Actual street*/
```

Fig. 3. Codul ce gestioneaza trimiterea de viteza la un interval de timp din client

– *Informatii esentiale despre un nou drum pe care intra un client*
Clientul trimite o cerere catre server pentru a afla informatii noi de fiecare data cand se observa o schimbare de strada. Server-ul va raspunde in conformitate cu datele salvate in baza de date.
– *Informatii aditionale*
Clientul poate trimite o cerere catre server pentru a afla informatii despre vreme, evenimente sportive, preturi pentru combustibili la statiile peco.

```c
else if(logged==1)
{
    if (strncmp(commandReceived, "Login: ", 7) == 0)
    {
        respond=strdup("You are already logged in!\n");
    }
    else if (strncmp(commandReceived, "Extra: ", 7) == 0) { ... }
    /* Option to select the cantity of information received:
    Extra 0- Speed information
    Extra 1- Actions&Objectives based on the street
    Extra 2- Actions&Objectives based on the neighbourhood_id
    Schema: "Extra: <level>" */

    else  if (strncmp(commandReceived, "Change: ", 8) == 0)
    {
        char* change=commandReceived+8;
        char *temp=malloc(sizeof(char)*2048);
        command_changeLocation(change, extra, temp);
        respond=strdup(temp);
    }
    /* API command_change(char*, int, char*) questions the database in order to get the amount information selected previously. It can take a new street, or the actual street.
    Schema: "Change: S:<nameOfTheStreet>" - to find out details about a certain street or
            "Change: N:<nameOfTheNeighbourdhood>" to find out details about a certain neighbourhood*/
```

Fig. 4. Partea de cod ce gestioneaza fluxul de informatii pe care serverul in trimite inapoi clientului

Raportare creaza o comunicare intre client si server de tipul scriere, unde server-ul primeste date pe care le va prelucra.

Sunt 2 situatii in care serverul prelucreaza datele: *informatia nu era cunoscuta* sau *informatia nu mai este valida.*

Daca server-ul primeste informatii pe care nu le avea stocate in baza de date, va incarca datele in baza de date si va anunta toti restul clientilor conectati.

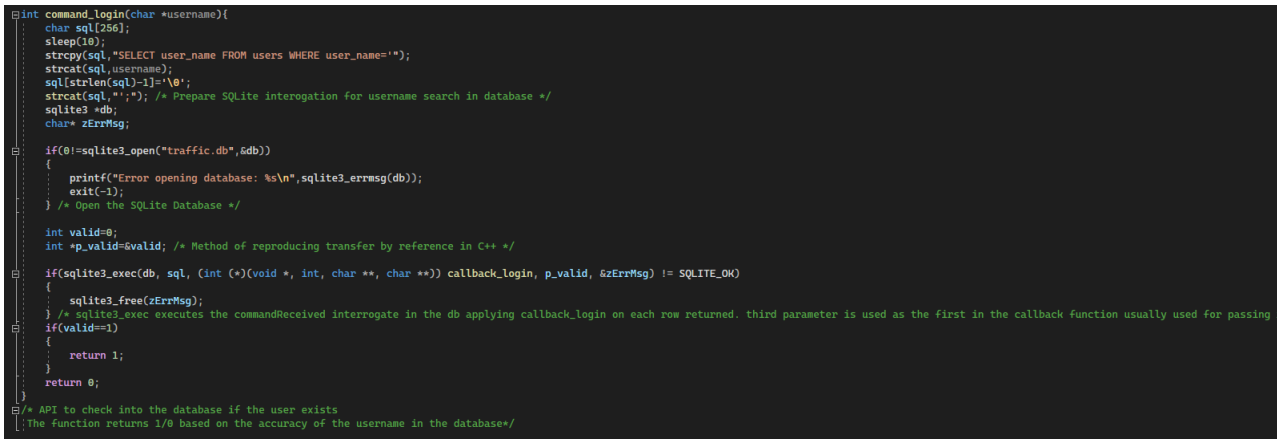
```
else  if (strncmp(commandReceived, "Change: ", 8) == 0)
{
    char* change=commandReceived+8;
    char *temp=malloc(sizeof(char)*2048);
    command_changeLocation(change, extra, temp);
    respond=strdup(temp);
}
/* API command_change(char*, int, char*) questions the database in order to get the amount information selected previously. It can take a new street, or the actual street.
Schema: "Change: S:<nameOfTheStreet>" - to find out details about a certain street or
        "Change: N:<nameOfTheNeighbourdhood>" to find out details about a certain neighbourhood*/

else if (strncmp(commandReceived, "Alert: ", 7) == 0)
{
    //using 'commandReceived' as a returning object because temp is not working properly
    char *temp=malloc(sizeof(char)*CHARSIZE);
    int spread=command_alert(commandReceived, temp);
    respond=strdup(commandReceived);
    //if spread==0, the alert will go to all the users connected at this moment
    if(spread==0)
    {
        //going through all the connected users
        for(int i=0;i<counter;++i)
        {
            if(connected[i]!=0 && i!=tdL.idThread)
            {
                int l = strlen(respond);
                if (write(client_list[i], &l, sizeof(int)) <= 0)
                {
                    printf("[thread %d] ", tdL.idThread);
                    perror("[thread] Can't write respond length.\n");
                }
                if (write(client_list[i], respond, l) <= 0)
                {
                    printf("[thread %d] ", tdL.idThread);
                    perror("[thread] Can't write message  to client.\n");
                }
            }
        }
        /* Iterate through all the other clients and send the alert to them */
    }
}
/* API commnad_allert(char*, char*) takes the alert and based on what exists in the database, returns different things.
If the alert already exists, the it will inform only this user that it is a known problem.
If the alert is new, all the users connected will be notified and the event will be stored in the database.
Schema: "Alert: T:<nameOfTheType> S:<nameOfTheSreet>"
ex: "Alert: T:Blocaj S:DA"*/
```

Fig. 5. Partea de cod ce gestioneaza alertele pe care serverul le primeste de la client si si trimite mai departe, daca este cazul, notificare catre restul clientilor conectati

Utilizatorul final are acces la urmatoare comenzi:

- Login: conectarea utilizatorului la contul lui [Fig. 6]
- Aditional: utilizatorul afla ce fel de informatii poate primi de la aplicatie:
  - 0: informatii vitale
  - 1: informatii aditionale referitoare la strada pe care se afla
  - 2: informatii aditionale referitoare la zona in care se afla
- Schimbare: anuntarea serverului ca utilizatorul schimba strada
- Cautare: cautarea unui loc/eveniment din baza de date [Fig. 7]
- Semnalizare: semnalizarea ca un eveniment (nu) are loc



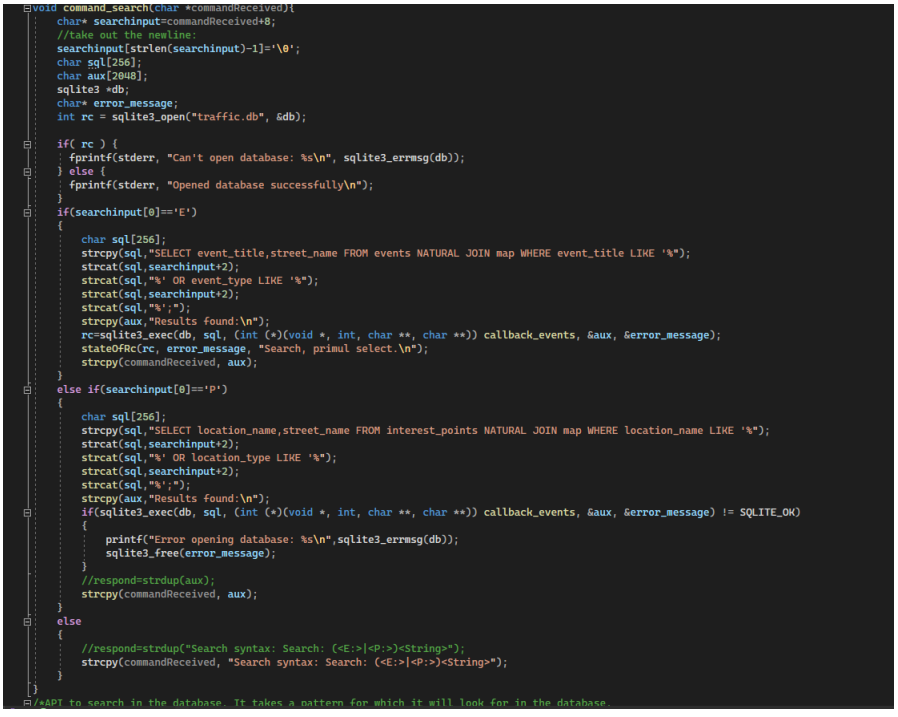
```
int command_login(char *username){
    char sql[256];
    sleep(10);
    strcpy(sql,"SELECT user_name FROM users WHERE user_name='");
    strcat(sql,username);
    sql[strlen(sql)-1]='\0';
    strcat(sql,"';"); /* Prepare SQLite interogation for username search in database */
    sqlite3 *db;
    char* zErrMsg;

    if(0!=sqlite3_open("traffic.db",&db))
    {
        printf("Error opening database: %s\n",sqlite3_errmsg(db));
        exit(-1);
    } /* Open the SQLite Database */

    int valid=0;
    int *p_valid=&valid; /* Method of reproducing transfer by reference in C++ */

    if(sqlite3_exec(db, sql, (int (*)(void *, int, char **, char **)) callback_login, p_valid, &zErrMsg) != SQLITE_OK)
    {
        sqlite3_free(zErrMsg);
    } /* sqlite3_exec executes the commandReceived interrogate in the db applying callback_login on each row returned. third parameter is used as the first in the callback function usually used for passing
    if(valid==1)
    {
        return 1;
    }
    return 0;
}
/* API to check into the database if the user exists
The function returns 1/0 based on the accuracy of the username in the database*/
```

Fig. 6. Partea de cod ce gestioneaza cererea de login de la client in server



```
void command_search(char *commandReceived){
    char* searchinput=commandReceived+8;
    //take out the newline:
    searchinput[strlen(searchinput)-1]='\0';
    char sql[256];
    char aux[2048];
    sqlite3 *db;
    char* error_message;
    int rc = sqlite3_open("traffic.db", &db);

    if( rc ) {
      fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
    } else {
      fprintf(stderr, "Opened database successfully\n");
    }
    if(searchinput[0]=='E')
    {
        char sql[256];
        strcpy(sql,"SELECT event_title,street_name FROM events NATURAL JOIN map WHERE event_title LIKE '%");
        strcat(sql,searchinput+2);
        strcat(sql,"%' OR event_type LIKE '%");
        strcat(sql,searchinput+2);
        strcat(sql,"%';");
        strcpy(aux,"Results found:\n");
        rc=sqlite3_exec(db, sql, (int (*)(void *, int, char **, char **)) callback_events, &aux, &error_message);
        stateOfRc(rc, error_message, "Search, primul select.\n");
        strcpy(commandReceived, aux);
    }
    else if(searchinput[0]=='P')
    {
        char sql[256];
        strcpy(sql,"SELECT location_name,street_name FROM interest_points NATURAL JOIN map WHERE location_name LIKE '%");
        strcat(sql,searchinput+2);
        strcat(sql,"%' OR location_type LIKE '%");
        strcat(sql,searchinput+2);
        strcat(sql,"%';");
        strcpy(aux,"Results found:\n");
        if(sqlite3_exec(db, sql, (int (*)(void *, int, char **, char **)) callback_events, &aux, &error_message) != SQLITE_OK)
        {
            printf("Error opening database: %s\n",sqlite3_errmsg(db));
            sqlite3_free(error_message);
        }
        //respond=strdup(aux);
        strcpy(commandReceived, aux);
    }
    else
    {
        //respond=strdup("Search syntax: Search: (<E:>|<P:>)<String>");
        strcpy(commandReceived, "Search syntax: Search: (<E:>|<P:>)<String>");
    }
}
/*API to search in the database. It takes a pattern for which it will look for in the database.
```

Fig. 7. Partea de cod ce gestioneaza cererea de login de la client in server

4.2 Interpretarea traficului

Pentru a transpune problema traficului in date ce pot fi prelucrate de aceasta aplicatie au fost luate urmatoarele decizii: se folosesc doar strazile din oras, cu posibilitatea de a grupa un numar de strazi in zona.

4.3 Scenarii de utilizare

Clientul se poate conecta la server doar daca cunoaste adresa si portul la care sa se conecteze. Ulterior, server-ul accepta cererea de conectare a clientului si creeaza un thread personal. Comunicarea intre server si client continua, fiind din acest punct o comunicare intre thread-ul prorpiu al acelui client facut de server si clientul respectiv.

Exemplu de alerta:

Un utilizator semnalizeaza un accident, thread-ul responsabil cu acest utlizator adauga accidentul in baza de date ca fiind un eveniment temporal. Ulterior, restul utlizatorilor vor fi anuntati despre accident.

Daca mai mult de 3 utilizatori semnalizarea ca accidentul nu mai este prezent, acesta va fi scos din baza de date. O data evenimentul scos, nu va mai aparea in urmatoarele interogari.

5 Concluzii

Aceasta aplicatie imbunatateste experienta de condus a soferilor dintr-un oras aglomerat insa nu este perfecta.

Cateva din lucrurile ce ar aduce imbunatatiri:

- O interfata grafica
- Posibilitatea de calculare a unei rute prin selectarea punctului de porninre si respectiv de oprire

References

- Informatii folosite in dezvoltarea proiectului (https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php)
- Cerinta proiectului (https://profs.info.uaic.ro/ computernetworks/ProiecteNet2022.php)

- Editor de imagine
  (https://app.diagrams.net/)
- Editor de text
  (https://www.overleaf.com/)