Documentația proiectului "Monitorizarea traficului"

Turcu Mihai-Ionuț
Anul II Grupa B3
2022-2023

Universitatea "Alexandru Ioan Cuza" din Iași
Facultatea de Informatica

Introducere

Aplicația "Monitorizarea traficului" are scopul de a forma o comunitate de utilizatori ce vor avea posibilitatea de a se ajuta reciproc în traficul urban si extraurban.

Utilizatorii își pot crea un cont pe care să îl configureze după preferințele proprii. Aceștia vor putea să aleagă în momentul creării contului dacă doresc să primească informații despre sport, vreme sau prețurile combustibilului de la benzinăriile din apropiere. Locația curentă a utilizatorilor va fi stocată în baza de date a server-ului, fiind actualizată de fiecare dată când aceștia își vor schimba direcția de deplasare.

În timpul folosirii aplicației, fiecare client va trimite automat viteza de deplasare la un interval de un minut, iar utilizatorul va avea posibilitatea de a transmite sistemului informații despre evenimente petrecute in trafic (accident, poliție, trafic aglomerat, blocaj etc.). Aceste informații vor fi trimise de server către fiecare client, menționandu-se și strada pe care are loc evenimentul. Astfel, șoferii se vor putea informa reciproc prin intermediul aplicației despre anumite evenimente petrcute in trafic.

Tehnologii utilizate

Protocolul de comunicare aflat la baza acestei aplicații este TCP ("Transmission Control Protocol").

TCP este un protocol folosit de aplicații care au nevoie de confirmare de primire a datelor. Efectuează o conectare virtuală full duplex între două puncte terminale, fiecare punct fiind definit de către o adresă IP și de către un port TCP.

Acest protocol detectează probleme precum pierderea pachetelor, dupliarea sau livrarea acestora într-o altă ordine și solicită retransmiterea pachetelor pierdute, rearanjează pachetele în ordine și ajută la minimizarea traficului din rețea în vederea reducerii apariției altor probleme.

Așadar, TCP este un protocol ideal pentru această aplicație, fiind un serviciu de încredere care garantează livrarea unui flux de date trimis de la o gazdă la alta. Livrarea corectă a informației este lucrul cel mai important în cadrul acestui proiect. Dacă nu am avea o siguranță asupra pachetelor primite, ar fi posibilă apariția unor erori precum: netrimiterea unui eveniment din trafic de către un client, trimiterea incompletă a informației de la server către client (de exemplu, doar jumătate din evenimentele sportive ce vor avea loc sunt primite de către clienți). Apariția unor astfel de erori nu va ajuta la decongestionarea traficului, făcând aplicația să nu ofere servicii de calitate utilizatorilor.

Avand în vedere faptul ca aplicația trebuie să servească utilizatorii in mod concurent, o tehnică optimă din punctul de vedere al memoriei este utilizarea firelor de execuție(thread-uri).

În informatică, un fir de execuție este cea mai mică secvență de instrucțiuni programate care poate fi gestionată independent de un planificator, care este de obicei o parte a sistemului de operare. Implementarea firelor de execuție și a proceselor diferă între sistemele de operare, dar în

majoritatea cazurilor un fir de execuție este o componentă a unui proces. Firele multiple ale unui anumit proces pot fi executate concomitent(multithreading), partajând resurse precum memoria. În special, firele de execuție ale unui proces împărtășesc codul său executabil, valorile variabilelor sale alocate dinamic și valorile variabilelor globale non thread-local în orice moment dat.

Multithreading-ul este capacitatea unui program sau a unui sistem de operare de a permite existența mai multor utilizatori simultan, fără a necesita mai multe copii ale programului care rulează pe computer. Fiecare cerere de utilizator pentru un program sau serviciu de sistem este urmărită ca un fir de execuție cu o identitate separată. Pe măsură ce programele funcționează în numele cererii inițiale de fir și sunt întrerupte de alte solicitări, starea de lucru a cererii inițiale este urmărită până la finalizarea lucrării.

Pentru stocarea datelor (străzi, nume de utilizatori, parole etc.) se folosește SQLite. SQLite este un sistem popular de gestionare a bazelor de date ce are urmatoarele avantaje:

- flexibilitatea;
- scrierea si citirea sunt mai eficiente într-o bază de date decât folosirea unor fișiere aflate pe disk;
- modul eficient de stocare a datelor in tabele;
- conținutul poate fi accesat si actualizat prin interogări SQL;
- actualizarea rapidă si ecifientă a conținutului;

Arhitectura aplicatiei

În diagrama de mai jos este prezentată detaliat arhitectura aplicației. Aplicatia va utiliza o singură bază de date ce va conține mai multe tabele. In figura de mai jos au fost folosite mai multe finguri reprezentative pentru o mai bună vizualizare.

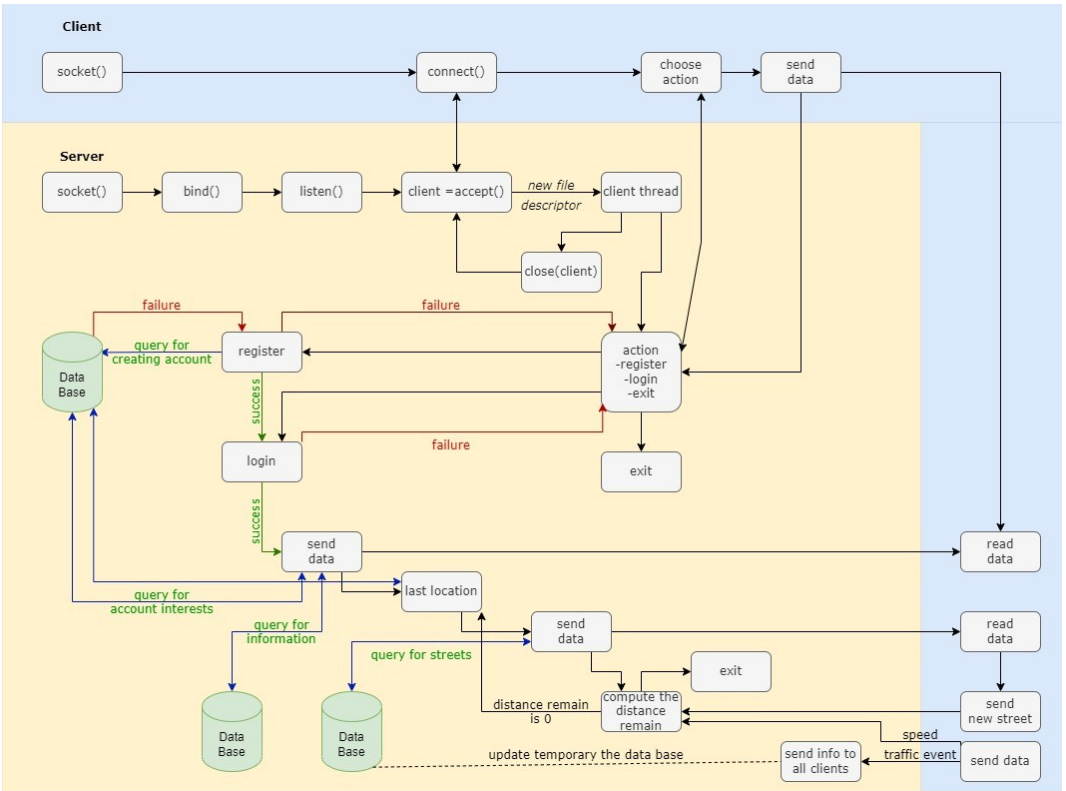


Figure 1: Diagrama aplicației

Tabelele ce se regăsesc in baza de date a server-ului sunt următoarele:

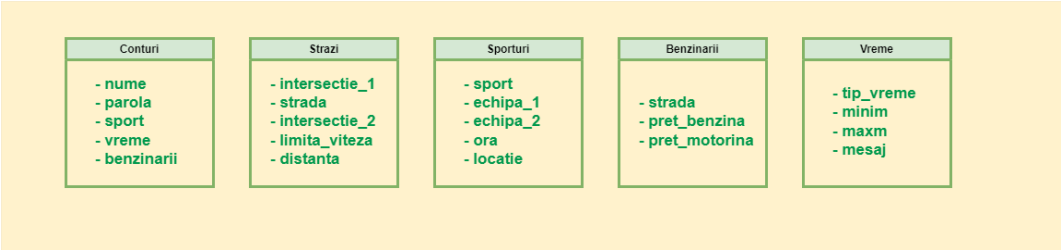


Figure 2: Baza de date a aplicației

Login & Register

Utilizatorii vor fi nevoiți să-și creeze și configureze propriul cont înainte de a putea folosi aplicația. Configurarea constă în setarea locației curente și alegerea tipurilor de informații pe care dorește să le primească.

Dacă utilizatorul deține deja un cont, nu-i rămâne decât să introducă numele de utilizator și parola pentru a se conecta.

Sport, Vreme & Prețuri la combustibil

Serverul stochează informații despre vreme, evenimente sportive și combustibilul de la benzinăriile din oraș în baza de date. Astfel, aceste informații vor fi extrase cu ușurință si trimise către utilizator. Aceste va primi informațiile despre lucrurile de care este interesat, datorită configurației contului său.

Viteza

Viteza este trimisă automat de client către server cu o freecvență de un minut. Este folosită o funcție ce returnează *random* o valoare dintr-un inverval specificat pentru a juca rolul unui "vitezometru".

Străzile

Străzile sunt stocate in baza de date, iar de fiecare dată cand un utilizator ajunge intr-o intersecție nouă se interoghează baza de date despre străzile indicente cu intersecția respectivă si vor fi trimise clientului. Clientul folosește o funcție ce selectează random o stradă pe care urmează să circule și informează serverul despre alegerea sa (lucru făcut pentru a permite clientului să joace rolul unei mașini care se deplasează). Serverul urmează să interogheze baza de date și să obțină distanța pana la următoarea intersecție, recalculând-o la fiecare reactualizare a vitezei utilizatorului.

Evenimente din trafic

Fiecare utilizator va avea posibilitatea de a raporta evenimente din trafic (poliție, accident, trafic aglomerat, blocaj etc.) Aceasta informație va fi trimisă automat de server către toți utilizatorii conectați, menționându-se și strada pe care are loc evenimentul.

Detalii de implementare

Atat serverul, cat si clientul stocheaza într-o structură informații esențiale despre utizilator și locația acestuia. Aceste informații vor ajuta la o comunicare logica.

```
67  void worker(int cl, int idThread)
68  {
69      sqlite3_stmt *res;
70      struct users{
71          char username[100];
72          char password[100];
73          char intersectie1[100];
74          char intersectie2[100];
75          char strada[100];
76          int viteza;
77          int limita_viteza;
78          int distanta_ramasa;
79          int vreme;
80          int sport;
81          int combustibil;
82      }user;
```

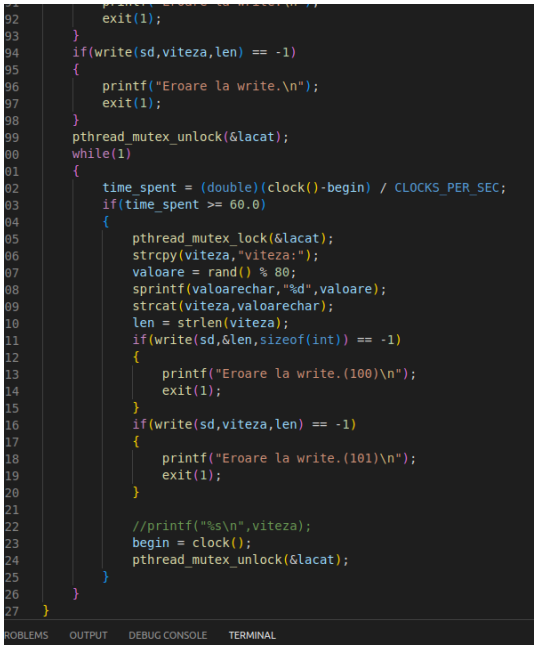
Figure 3: Structură folosită de server

Imediat dupa ce utilizatorul se conecteaza, clientul creeaza un nou Thread care trimite catre server o viteza random intre 0 si 80 km/h. Acest thread cronometreaza secundele si trimite o noua viteza o data la un minut. [Figura 4 si Figura 5]

```
70  }
71
72  void *speed( void * arg )
73  {
74      int sd;
75      sd = (int) arg;
76      clock_t begin;
77      double time_spent;
78      unsigned int i;
79      begin = clock();
80      char viteza[20];
81      char valoarechar[10];
82      int valoare;
83      strcpy(viteza,"viteza:");
84      valoare = rand() % 80;
85      sprintf(valoarechar,"%d",valoare);
86      strcat(viteza,valoarechar);
87      pthread_mutex_lock(&lacat);
88      len = strlen(viteza);
89      if(write(sd,&len,sizeof(int)) == -1)
90      {
91          printf("Eroare la write.\n");
92          exit(1);
93      }
94      if(write(sd,viteza,len) == -1)
95      {
96          printf("Eroare la write.\n");
97          exit(1);
98      }
99      pthread_mutex_unlock(&lacat);
00      while(1)
01      {
02          time_spent = (double)(clock()-begin) / CLOCKS_PER_SEC;
03          if(time_spent >= 60.0)
04          {
05              pthread_mutex_lock(&lacat);
06              strcpy(viteza,"viteza:");
ROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
```

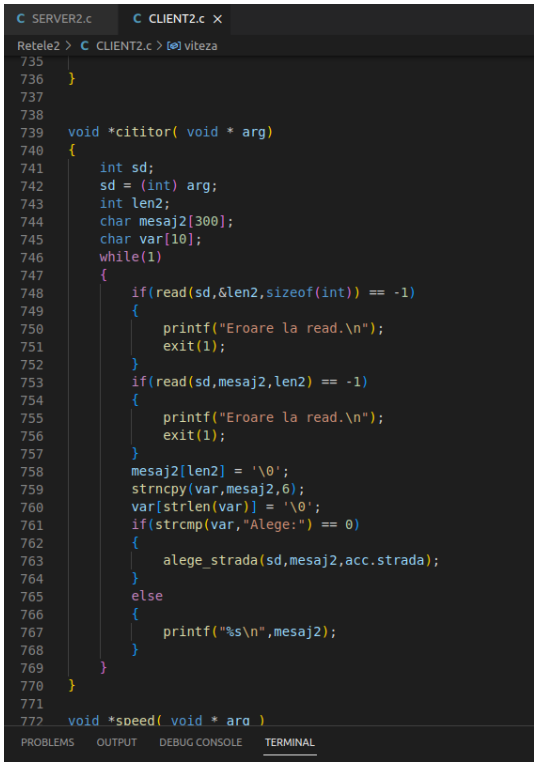
Figure 4: Thread care trimite viteza catre server I

Dupa conectare, clientul creeaza un thread (cititor) care va astepta sa primeasca pachete de la server. Evenimentele petrecute in trafic trebuie sa fie primite de catre toti clientii imediat dupa raportarea acestora. Astfel fiecare client trebuie sa fie pregatit tot timpul sa citeasca un pachet, iar thread-ul caruia i se asociaza functia cititor ajuta in acest scop. [Figura 6]



```
92          exit(1);
93      }
94      if(write(sd,viteza,len) == -1)
95      {
96          printf("Eroare la write.\n");
97          exit(1);
98      }
99      pthread_mutex_unlock(&lacat);
00      while(1)
01      {
02          time_spent = (double)(clock()-begin) / CLOCKS_PER_SEC;
03          if(time_spent >= 60.0)
04          {
05              pthread_mutex_lock(&lacat);
06              strcpy(viteza,"viteza:");
07              valoare = rand() % 80;
08              sprintf(valoarechar,"%d",valoare);
09              strcat(viteza,valoarechar);
10              len = strlen(viteza);
11              if(write(sd,&len,sizeof(int)) == -1)
12              {
13                  printf("Eroare la write.(100)\n");
14                  exit(1);
15              }
16              if(write(sd,viteza,len) == -1)
17              {
18                  printf("Eroare la write.(101)\n");
19                  exit(1);
20              }
21
22              //printf("%s\n",viteza);
23              begin = clock();
24              pthread_mutex_unlock(&lacat);
25          }
26      }
27  }
ROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
```

Figure 5: Thread care trimite viteza catre server II



```
C SERVER2.c   C CLIENT2.c ×
Retele2 > C CLIENT2.c > viteza
735
736 }
737
738
739 void *cititor( void * arg)
740 {
741     int sd;
742     sd = (int) arg;
743     int len2;
744     char mesaj2[300];
745     char var[10];
746     while(1)
747     {
748         if(read(sd,&len2,sizeof(int)) == -1)
749         {
750             printf("Eroare la read.\n");
751             exit(1);
752         }
753         if(read(sd,mesaj2,len2) == -1)
754         {
755             printf("Eroare la read.\n");
756             exit(1);
757         }
758         mesaj2[len2] = '\0';
759         strncpy(var,mesaj2,6);
760         var[strlen(var)] = '\0';
761         if(strcmp(var,"Alege:") == 0)
762         {
763             alege_strada(sd,mesaj2,acc.strada);
764         }
765         else
766         {
767             printf("%s\n",mesaj2);
768         }
769     }
770 }
771
772 void *speed( void * arg )
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
```

Figure 6: Functia executata de un thread creat dupa conectare

Pentru a evita coliziunile de pachete am folosit declarat un vector de mutex-uri, fiecare thread asociat unui client avand propriul lacat pe care il va bloca inainte de orice scriere catre client si debloca imediat dupa ce pachetul a fost trimis. [Figura 7 si Figura 8]

```
    /* Structura in care retinem informatiile thread-urilor*/
typedef struct {
    pthread_t idThread; // id-ul thread-ului
    int thCount; // nr de conexiuni servite
}Thread;

Thread *threadsPool; // un array de structuri Thread

pthread_mutex_t lacate[NUMBER_THREADS] = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mlock = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t lock_db = PTHREAD_MUTEX_INITIALIZER;
pthread_t informatii;

sqlite3 *db;
```

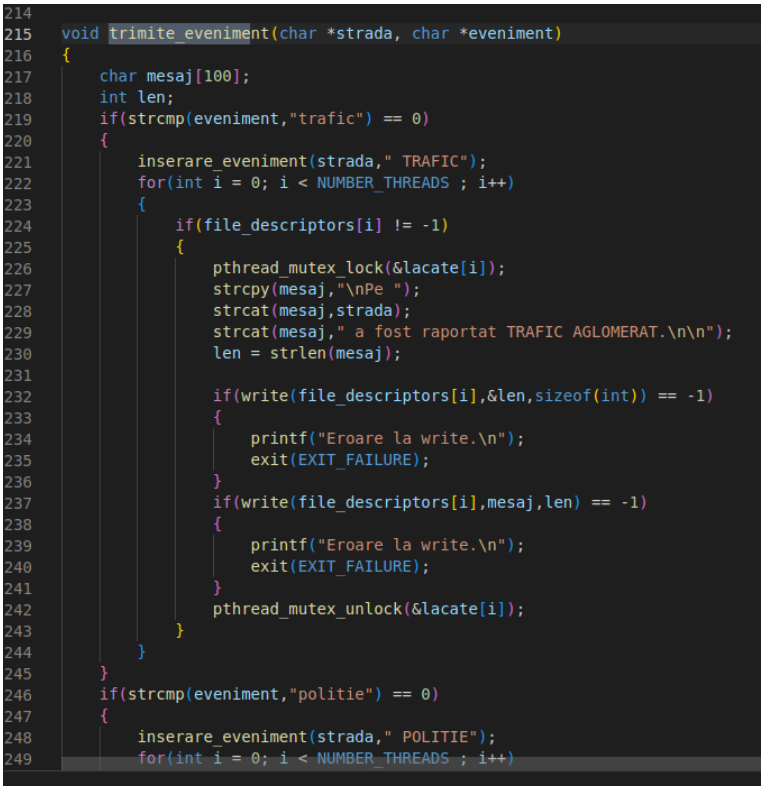
Figure 7: Creare de lacate pentru fiecare client

```
        pthread_mutex_unlock(&lacate[idThread]);
        reusit = 1;
    }
    else
    {
        loop++;
        len = strlen("Eroare la crearea contului.");
        pthread_mutex_lock(&lacate[idThread]);
        if(write(cl,&len,sizeof(int)) == -1)
        {
            printf("Eroare la write(622).\n");
            exit(EXIT_FAILURE);
        }
        if(write(cl,"Eroare la crearea contului.",len) == -1)
        {
            printf("Eroare la write(627).\n");
            exit(EXIT_FAILURE);
        }
        pthread_mutex_unlock(&lacate[idThread]);
    }
    pthread_mutex_unlock(&lock_db);
```

Figure 8: Utilizarea lacatelor la scriere

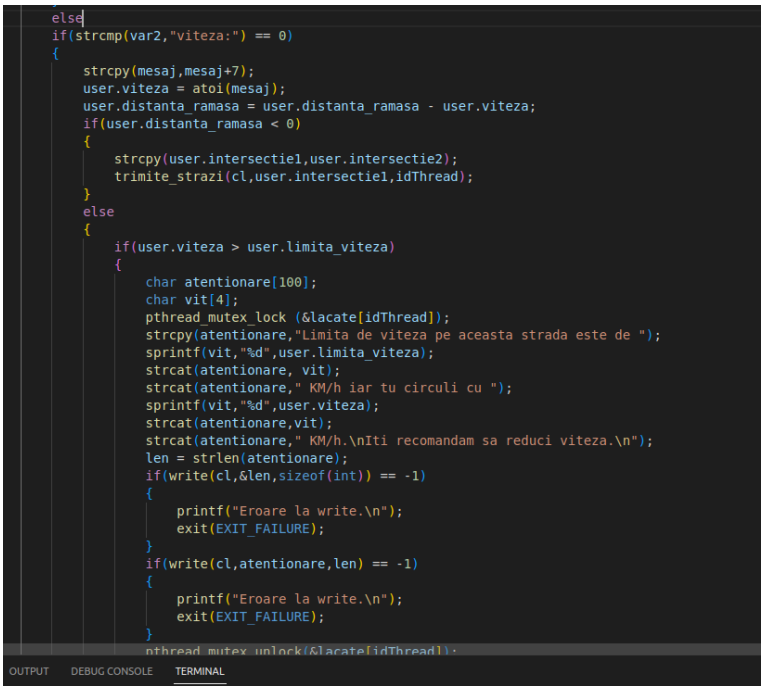
Toti file descriptorii clientilor vor fi adaugati intr-un vector in momentul conectarii. In momentul raportarii unui eveniment vectorul va fi parcurs trimitand un mesaj de informare tuturor clientilor conectati. [Figura 9]

De fiecare data cand un client trimite viteza cu care circula, serverul calculeaza distanta ramasa pana la urmatoarea intersectie. Daca distanta este mai mica decat 0 inseamna ca utilizatorul a parcurs strada curenta si trebuie sa aleaga o strada noua pe care se va deplasa. [Figura 10]



```
214
215 void trimite_eveniment(char *strada, char *eveniment)
216 {
217     char mesaj[100];
218     int len;
219     if(strcmp(eveniment,"trafic") == 0)
220     {
221         inserare_eveniment(strada," TRAFIC");
222         for(int i = 0; i < NUMBER_THREADS ; i++)
223         {
224             if(file_descriptors[i] != -1)
225             {
226                 pthread_mutex_lock(&lacate[i]);
227                 strcpy(mesaj,"\nPe ");
228                 strcat(mesaj,strada);
229                 strcat(mesaj," a fost raportat TRAFIC AGLOMERAT.\n\n");
230                 len = strlen(mesaj);
231
232                 if(write(file_descriptors[i],&len,sizeof(int)) == -1)
233                 {
234                     printf("Eroare la write.\n");
235                     exit(EXIT_FAILURE);
236                 }
237                 if(write(file_descriptors[i],mesaj,len) == -1)
238                 {
239                     printf("Eroare la write.\n");
240                     exit(EXIT_FAILURE);
241                 }
242                 pthread_mutex_unlock(&lacate[i]);
243             }
244         }
245     }
246     if(strcmp(eveniment,"politie") == 0)
247     {
248         inserare_eveniment(strada," POLITIE");
249         for(int i = 0; i < NUMBER_THREADS ; i++)
```

Figure 9: Utilizarea lacatelor la scriere



```
else
if(strcmp(var2,"viteza:") == 0)
{
    strcpy(mesaj,mesaj+7);
    user.viteza = atoi(mesaj);
    user.distanta_ramasa = user.distanta_ramasa - user.viteza;
    if(user.distanta_ramasa < 0)
    {
        strcpy(user.intersectie1,user.intersectie2);
        trimite_strazi(cl,user.intersectie1,idThread);
    }
    else
    {
        if(user.viteza > user.limita_viteza)
        {
            char atentionare[100];
            char vit[4];
            pthread_mutex_lock (&lacate[idThread]);
            strcpy(atentionare,"Limita de viteza pe aceasta strada este de ");
            sprintf(vit,"%d",user.limita_viteza);
            strcat(atentionare, vit);
            strcat(atentionare," KM/h iar tu circuli cu ");
            sprintf(vit,"%d",user.viteza);
            strcat(atentionare,vit);
            strcat(atentionare," KM/h.\nIti recomandam sa reduci viteza.\n");
            len = strlen(atentionare);
            if(write(cl,&len,sizeof(int)) == -1)
            {
                printf("Eroare la write.\n");
                exit(EXIT_FAILURE);
            }
            if(write(cl,atentionare,len) == -1)
            {
                printf("Eroare la write.\n");
                exit(EXIT_FAILURE);
            }
            pthread_mutex_unlock(&lacate[idThread]);
```

OUTPUT DEBUG CONSOLE TERMINAL

Figure 10: Calcularea distantei pana la urmatoarea intersectie

Concluzii

Proiectul ar putea fi îmbunătățit astfel:

- calcularea drumului optim dintre locația de plecare a utilizatorului și destinația dorită de acesta;
- utilizatorul să aibă posibilitatea de reconfigurare a contului;
- actualizarea temporară a bazei de date în scopul unui eveniment transmis de către un client (limita de viteză să fie mai mică decât cea obișnuită pentru o perioadă de timp, urmând să redevină în starea ei inițială);

Bibliografie

Bibliografie utilizată în scopul realizării acestei documentații:

https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php

https://www.sqlite.org/aff_short.html

https://www.w3schools.blog/advantages-sqlite

https://ro.wikipedia.org/wiki/Transmission_Control_Protocol

https://www.techtarget.com/whatis/definition/multithreading

https://en.wikipedia.org/wiki/Thread_(computing)