Monitorizarea traficului(A)

Damian Alexandru, grupa B5, anul 2

Facultatea de informatica Iasi

Keywords: TCP · client-server · threads · C · C++

1 Introducere

Motivatie Am ales sa fac acest proiect deoarece am vrut sa aprofundez conceptele de retelistica. Consider ca acest proiect ma va ajuta sa inteleg mai bine cum sunt implementate sistemele de gestiune a traficului, sau aplicatiile folosite pentru navigare.

Voi realiza aplicatia in limbajul C si C++ cu o interfata grafica pentru client realizata in Qtcreator.

2 Tehnologiile utilizate

In acest proiect voi folosi protocolul TCP/IP deoarece vreau sa fiu sigur ca toti clientii vor primi notificarile pentru evenimente, accidente din zonele in care se afla. Prin utilizarea protocolului TCP/IP, clientul va crea o conexiune stabila cu serverul, astfel transferul de informatii va fi garantat.

Pentru a permite accesul mai multor clienti la server in acelasi timp, voi crea un server TCP concurent. Pentru fiecare client conectat, voi crea un nou thread in server.

De asemenea, la proiect ma voi folosi si de Qtcreator pentru a realiza interfata clientului.

3 Arhitectura aplicatiei

Serverul va astepta conexiuni din partea clientilor. Pentru fiecare client, serverul va obtine 2 file-descriptori prin care va comunica cu acel client, si va crea un thread doar pentru acel client. Deci pentru fiecare client, in server va exista cate un thread.

Primul fd, va fi pentru comunicarea datelor de logare/inregistrare, si comunicarea optiunilor si incidentelor trimise de client catre server. Al doilea fd, va

fi folosit pentru a realiza broadcastul si pentru a citi locatia clientului. Cand un client va trimite o informatie despre un accident, serverul va inregistra aceasta informatie, si va seta o variabila accesibila fiecarui thread astfel incat in fiecare thread va sti ca va trebui sa trimita informatia clientului sau.

Drumul unui client va fi stabilit folosind un algoritm dfs. Punctul de start, destinatia si viteza vor fi alese aleator, iar viteza se va actualiza si locatia se vor actualiza o data la 2 secunde. Orice locatie trimisa intre server si client este de forma "STRADA NUMAR".

Pentru a simula deplasarera unei masini, in client voi calcula distanta parcursa de client in 2 secunde, la viteza initiala. Din distanta totala a strazii, voi scadea aceasta distanta. Cand un client trimite locatia catre server, server-ul se va uita la strazile care au o legatura cu strada clientului si va trimite evenimentele aferente acelor strazi.

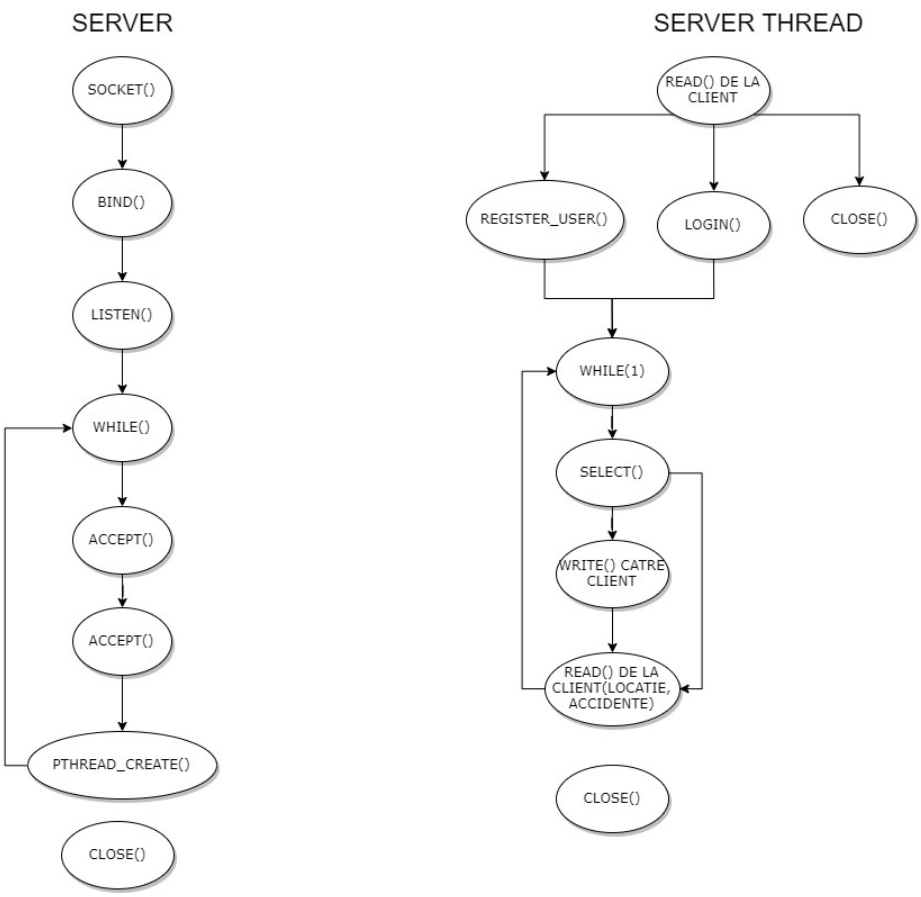

Fig. 1: Diagrama server

Clientul va avea un main thread si un listening thread si 2 socketi. Pe primul socket va comunica cu serverul datele de logare, si va introduce introduce date in caz de accident. Al doilea socket, va trimite date la server despre locatia si viteza clientului, si va primi de la server informatii despre limita de viteza, incidente (maracte de alti utilizatori) si evenimentele din acea zona.

Pentru a actualiza interfata clientului, ma voi folosi de sistemul SIGNAL - SLOT din QT. Astfel, cand voi primi o informatie noua in thread-ul de ascultare (listening thread) voi emite un semnal care va fi receptionat in client, unde va fi actualizata interfata.

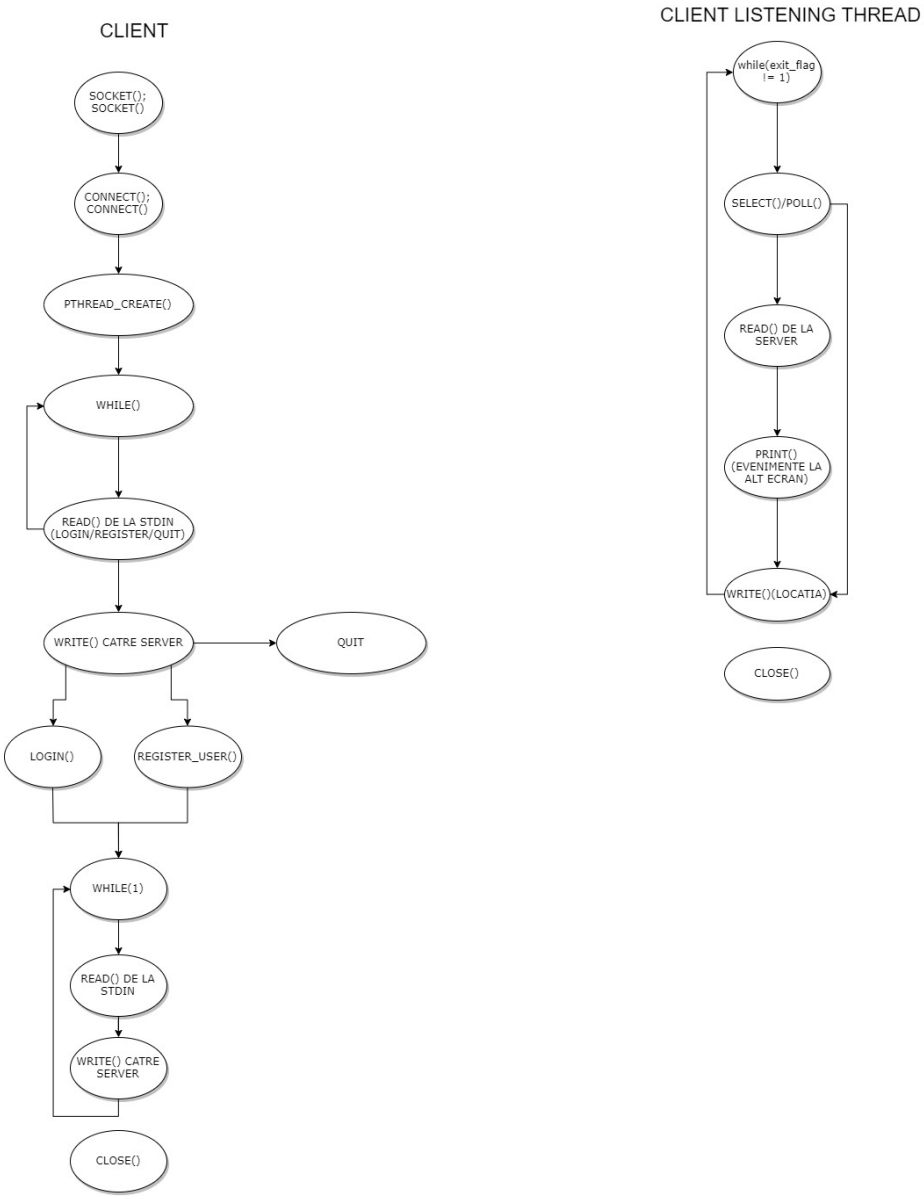

Fig. 2: Diagrama client

Clientul va avea un loc in care poate introduce "accident" sau diferite comenzi. De asemenea va avea si o lista care va fi actualizata cu evenimentele din jurul sau in timp real.

Pentru a simula strazile unui oras, voi folosi un graf. Acesta va fi retinut sub forma unui vector de muchii. Clientii se vor deplasa de la un nod x la un nod y. Aceste noduri vor fi alese aleator la inceput, iar drumul va fi ales folosind un dfs. Zonele cu restrictie de viteza, evenimentele si accidentele vor fi retinute de server intr-un vector de structuri, iar locatia lor va fi retinuta ca "Nume_strada numar_de_pe_strada".

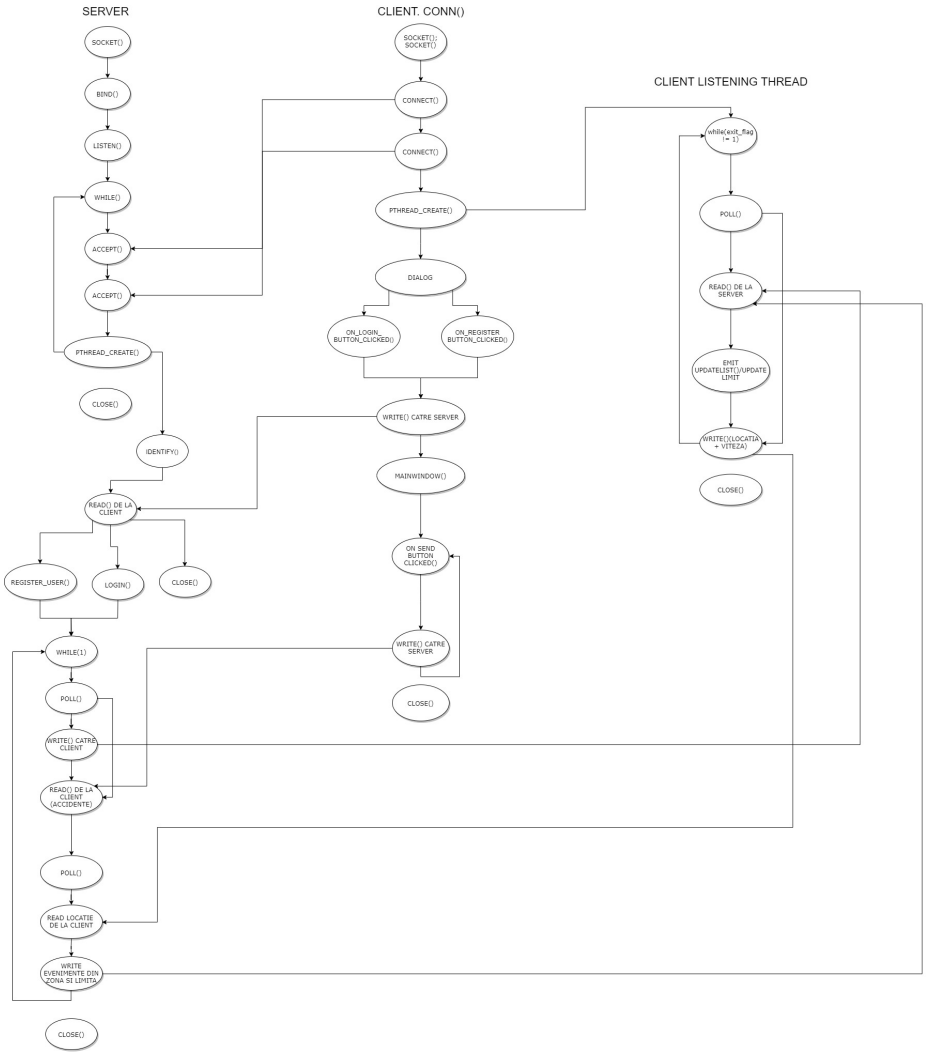

Fig. 3: Diagrama generala a aplicatiei

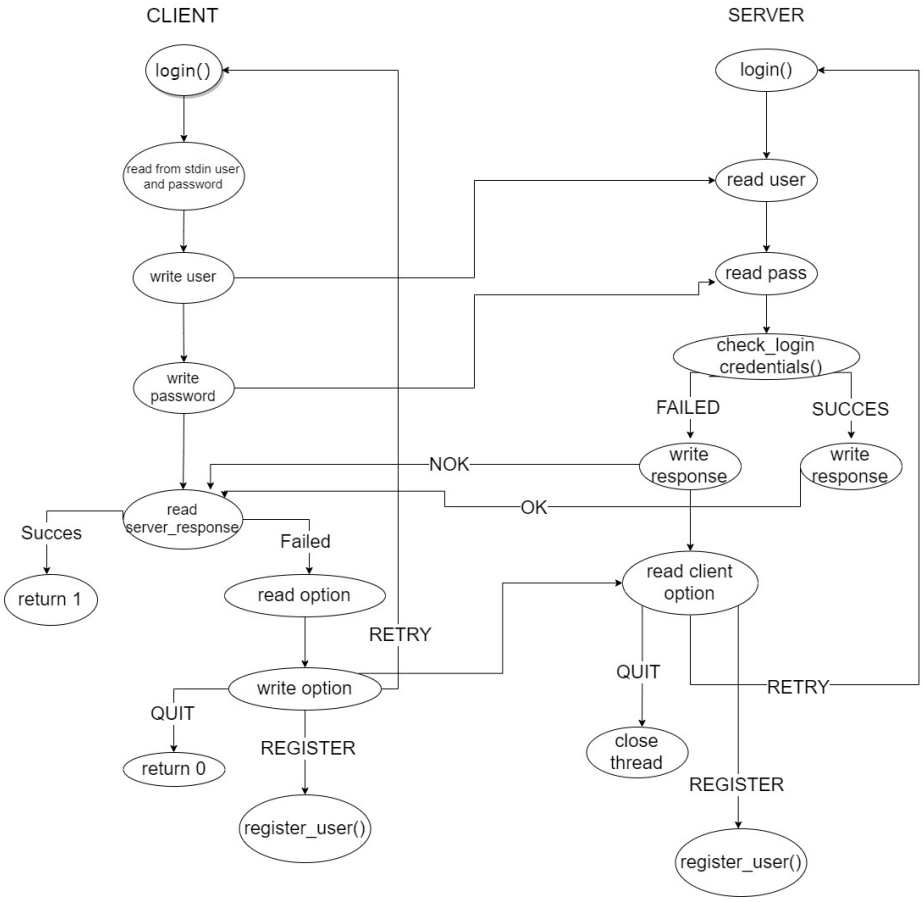

Fig. 4: Functia login

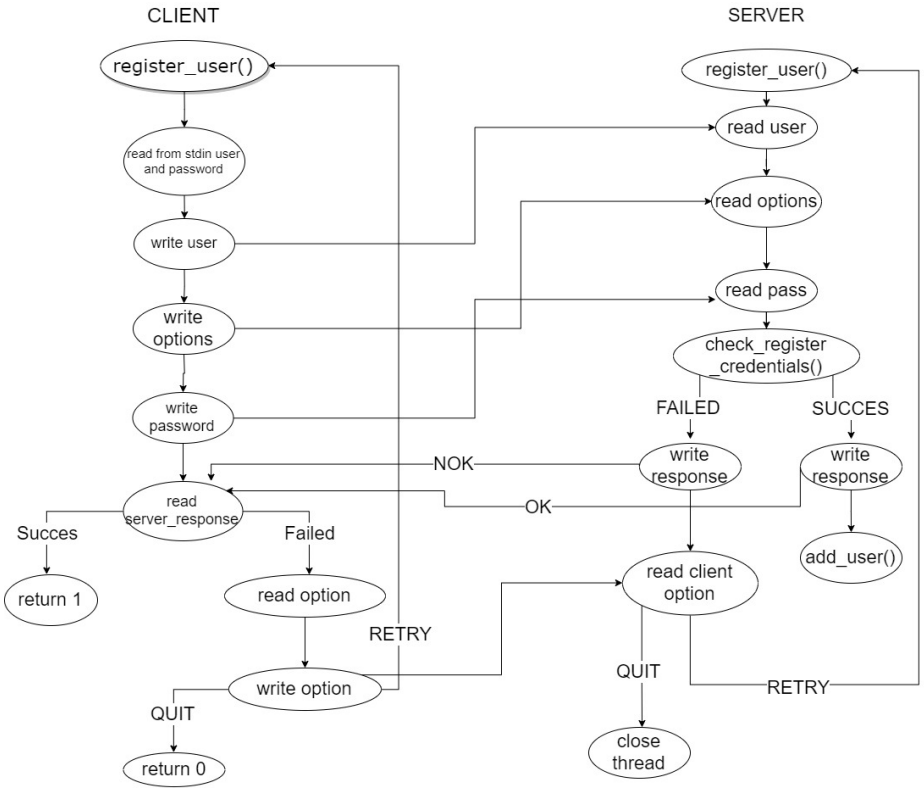

Fig. 5: Functia register

4 Detalii de implementare

Clientul va avea de ales intre register, login si quit. Daca se va inregistra, acesta va trebui sa isi aleaga si evenimentele pe care vrea sa le primeasca(informatii despre vreme, evenimente sportive, preturi pentru combustibili la statiile peco). Dupa ce conectarea s-a realizat cu succes, clientul va putea sa introduca accident sau inchide. Daca introduce accident locatia va fi luata automat si transimsa catre server.

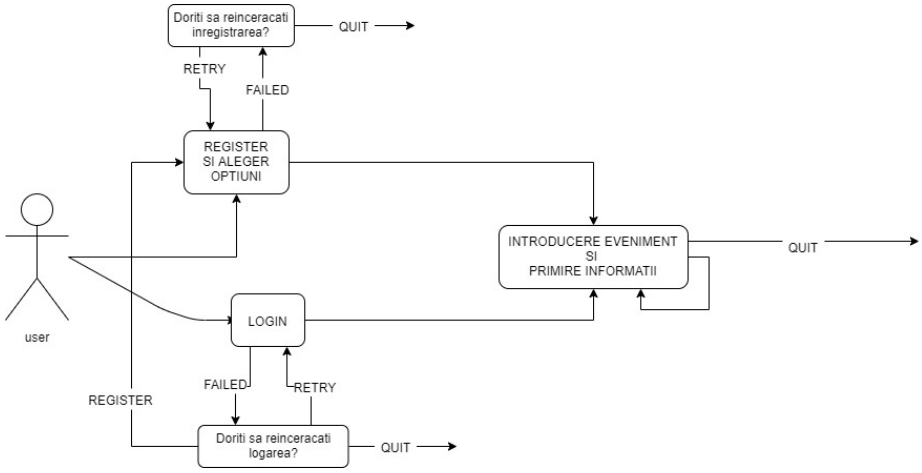

Fig. 6: Cum va folosi clientul aplicatia

Functia de login din server

```
1 int login(int sd, int id, char options[4]){
2     char credentials[100];
3     if(read(sd, credentials, 100) < 0){
4         perror("Eroare la citire mesaj");
5         return -1;
6     }
7
8
9     /// check ///
10    int check_code;
11    if(strlen(credentials) <= 2){
12        check_code = 0;
13    }
14    else check_code =
          check_login_credentials(credentials, options);
15    if(check_code == 0){
16        if(write(sd, "NOT OK", 7) < 0){
17            perror("Eroare la scriere mesaj");
18            return -1;
19        }
20        printf("[thread \%d] Clientul nu s-a logat cu
              succes!\n", id);
21        return 0;
22    }else if(check_code == 1){
23        if(write(sd, "OK", 3) < 0){
```

```
24              perror("Eroare la scriere mesaj");
25              return -1;
26          }
27          printf("[thread \%d] Clientul s-a logat cu
                succes!\n", id);
28          return 1;
29      }
30      return 0;
31  }
```

Functia de login din client

```
1  bool Client::conn(){//the listening thread will start
       after the connection was established
2      this->listen_socket = socket(AF_INET, SOCK_STREAM,
           0);
3      this->info_socket = socket(AF_INET, SOCK_STREAM, 0);
4      struct sockaddr_in server;
5
6      server.sin_addr.s_addr = inet_addr("127.0.0.1");
7      server.sin_port = htons(2038);
8      server.sin_family = AF_INET;
9
10     int conn_code = ::connect(this->info_socket, (struct
           sockaddr*)&server, sizeof(struct sockaddr));
11     int conn_code2 = ::connect(this->listen_socket,
           (struct sockaddr*)&server, sizeof(struct
           sockaddr));
12
13     if(conn_code == -1 || conn_code2 == -1){
14         ui->listWidget->addItem("Eroare la stabilire
               conexiune!");
15         return false;
16     }
17
18     form = new Dialog(this, info_socket);
19     int connect = form->exec();
20     if(connect == 1){
21         ui->listWidget->addItem("Conectare reusita!");
22         ui->listWidget->addItem("Bine ati venit!");
23         ui->listWidget->addItem("Aici veti primi
               informatii relevante locatiei curente.");
24         lThread->setDescriptor(listen_socket);
25         lThread->start();
26         return true;
```

```
27      }
28      return false;
29
30 }
```

Functia register user din server

```
1 void register_user(int sd, int id){
2     /// check ///
3     while(1){
4         char user[INPUT_SIZE];
5         char pass[INPUT_SIZE];
6         read(sd, user, INPUT_SIZE);
7         read(sd, pass, INPUT_SIZE);
8         int check_code =
              check_register_credentials(user, pass);
9         printf("code = \%d\n", check_code);
10        if(check_code == 1){
11            write(sd, "NOT OK", 7);
12            //vezi ce a trimis clientul
13            char option[100];
14            read(sd, option, 100);
15            if(strcmp(option, "quit") == 0){
16                check_code = -1;
17                break;
18            }else if(strcmp(option, "retry") == 0){
19                continue; //bucla while va merge inca o
                      data
20            }
21        }else if(check_code == 0){//add user
22            add_user(user, pass);
23            printf("[thread \%d]Clientul \%s s-a
                  inregistrat cu succes!\n", id, user);
24            write(sd, "OK", 3);
25            break;
26        }
27    }
28 }
```

Functia check login credentials din server

```
1 int check_login_credentials(char*credentials, char
    options[4]){// -1 daca e eroare, 1 daca am gasit user
    + pass, 0 altfel
2     char user[50], pass[50];
```

```
3       int k = 0, m = 0;
4       while(credentials[k] != '&'){
5           user[k] = credentials[k];
6           k++;
7       }
8       user[k] = '\0';
9
10      while(credentials[k] == '&'){
11          k++;
12      }
13      while(credentials[k] != '&' && k <
            strlen(credentials)){
14          pass[m++] = credentials[k++];
15      }
16      pass[m] = '\0';
17
18      int fd = open("useri.txt", O_RDWR);
19      if(fd < 0){
20          perror("Eroare la deschidere fisier");
21          return -1;
22      }
23      int found = 0;
24      k = 0;
25      char word1[110];
26
27      while(!found){//parcurg fisierul
28          unsigned char ch;
29          int codr = read(fd, &ch, 1);
30          if(codr == 0){  //end of file
31              break;
32          }else if(codr == -1){
33              perror("Eroare la citire");
34              close(fd);
35              return -1;
36          }
37          if(ch != ' ' && ch != '\n'){//obtinem user-ul de
                pe o linie
38              word1[k++] = ch;
39          }
40          if(ch == ' '){
41              word1[k] = '\0';
42              if(strcmp(word1, user) != 0){//daca e un
                    user diferit, sarim linia
43                  while(ch != '\n'){//parcurgem linia pana
                        la sfarsit
```

```
44                      read(fd, &ch, 1);
45                  }
46                  k = 0;
47              }else{ // daca am gasit user-ul verificam
                    parola
48                  int t = 0;
49                  char password[50];
50
51                  read(fd, &ch, 1);
52                  password[t++] = ch;
53                  while(ch != ' '){//obtinem parola
54                      read(fd, &ch, 1);
55                      password[t++] = ch;
56                  }
57                  password[t-1] = '\0';
58
59                  if(strcmp(pass, password) ==
                        0){//verificam parola
60                      //daca parola este buna, vom citi
                            optiunile
61
62                      for(int o = 0; o < 3; o++){
63                          read(fd, &ch, 1);
64                          options[o] = ch;
65                      }
66                      options[3] = '\0';
67                      close(fd);
68                      return 1;
69                  }
70                  while(ch != '\n'){//parcurgem linia pana
                        la sfarsit
71                      read(fd, &ch, 1);
72                  }
73                  k = 0;
74              }
75          }
76      }
77      close(fd);
78      return found;
79  }
```

Functia care obtine evenimentele din zona unei locatii

```
1 char* get_events(char*street_name, int number_on_street,
    int event_sent[100], char options[4]){
```

```
2     int found = 0, ref_number;
3     //obtinere indicele strazii clientului din vectorul
          strada[]
4     for(int i = 0; i < nr_strazi && found == 0; i++){
5         if(strcmp(strada[i].nume, street_name) == 0
6             && number_on_street >=
                  strada[i].nr_locuinte[0]
7             && number_on_street <=
                  strada[i].nr_locuinte[1]){
8                 found = 1;
9                 ref_number = i;
10            }
11    }
12    char*events = (char*)malloc(200);
13    events[0] = '\0';
14    strcat(events, "limita:");
15    strcat(events, itoa(strada[ref_number].limita));
16    strcat(events, "&&");
17    int k;
18    char**neighbours = get_neighbours(ref_number, &k);
19    for(int i = 0; i < total_events; i++){
20        for(int j = 0; j < k; j++){//verifica vecinii
21            if(strcmp(event[i].street, neighbours[j]) ==
                  0 && event_sent[i] == 0){//caut un
                  eveniment netrimis situat in apropiere
22
23                if(event[i].type[0] == 1 && options[0]
                      == '1'){//vreme
24                    strcat(events, "Vreme:");
25                    strcat(events, event[i].message);
26                    strcat(events, " pe strada ");
27                    strcat(events, event[i].street);
28                    if(event[i].number != -1){
29                        strcat(events, " nr. ");
30                        strcat(events,
                              itoa(event[i].number));
31                    }
32                    strcat(events, "&&");
33                    event_sent[i] = 1;
34                }else if(event[i].type[1] == 1 &&
                      options[1] == '1'){//sport
35                    strcat(events, "Sport:");
36                    strcat(events, event[i].message);
37                    strcat(events, " pe strada ");
38
```

```
39                      strcat(events, event[i].street);
40                      if(event[i].number != -1){
41                          strcat(events, " nr. ");
42                          strcat(events,
                                itoa(event[i].number));
43                      }
44                      strcat(events, "&&");
45                      event_sent[i] = 1;
46                  }else if(event[i].type[2] == 1
                        &&options[2] == '1'){//pret
47                      strcat(events, "Pret statie peco:");
48                      strcat(events, event[i].message);
49                      strcat(events, " pe strada ");
50
51                      strcat(events, event[i].street);
52                      if(event[i].number != -1){
53                          strcat(events, " nr. ");
54                          strcat(events,
                                itoa(event[i].number));
55                      }
56                      strcat(events, "&&");
57                      event_sent[i] = 1;
58                  }
59              }
60          }
61      }
62      return events;
63  }
```

Un thread din server

```
1  void* routine(void*arg){
2      thData thread = *((thData*)arg);
3      int sd = thread.client_fd;
4      printf("[thread \%d]Thread inceput.\n",
           thread.thread_id);
5      char options[3];
6      int idf = identify(sd, thread.thread_id, options);
7      if(idf == -1){
8          printf("[thread \%d] Inchidere thread.\n",
               thread.thread_id);
9          close((uintptr_t)arg);
10         return NULL;
11     }
12     char msg[100];
```

```
13      int broadcast_descriptor = thread.broadcast_fd, quit
           = 0;
14      int nr_accidente_din_thread = 0;
15      int client_speed = 0;
16      struct pollfd fds[1];
17      memset(fds, 0, sizeof(fds));
18      fds[0].fd = sd;
19      fds[0].events = POLLIN;
20
21      struct pollfd location_fd[1];
22      memset(location_fd, 0, sizeof(location_fd));
23      location_fd[0].fd = broadcast_descriptor;
24      location_fd[0].events = POLLIN;
25
26
27      int time = 2000;//2 seconds;
28      int event_sent[100];
29      for(int i = 0 ; i < 100; i++){
30          event_sent[i] = 0;
31      }
32      while(quit == 0){
33          printf("[thread \%d] Stau si ascult evenimente
               de la client\n", thread.thread_id);
34
35          int poll_code = poll(fds, 1, time); //al doilea
               parametru e folosit pentru a spune cate
               elemente are multimea de descriptori
36          if(poll_code < 0){
37              perror("Eroare la poll");
38              break;
39          }
40          if(poll_code == 0){//timeout
41              printf("[thread \%d]accident poll
                   timedout\n", thread.thread_id);
42          }
43          else {//s-a intamplat ceva pe socket
44              memset(msg, 0, 100);
45              int codr = read(fds[0].fd, msg, 100);
46              if(codr < 0){
47                  if(errno != EWOULDBLOCK){
48                      perror("Eroare la citire din
                           socket");
49                      break;
50                  }
51              }else if(codr == 0){
```

```
52                  printf("[thread \%d] Conexiune inchisa
                        de catre client\n", thread.thread_id);
53                  break;
54              }
55
56
57              printf("[thread \%d] am primit de la client
                    mesajul >\%s<\n", thread.thread_id, msg);
58              //mesajele primite de la client catre server
                    tratate aici sunt de tipul fi de tipul
59              // "accident&&nume_strada nr_de_pe_strada
60              if(strcmp(msg, "quit") == 0){
61                  quit = 1;
62                  break;
63              }else{
64                  char*x = strtok(msg, "&&");
65                  x = strtok(NULL, "&&");
66                  accidente[nr_total_accidente].locatie =
                        (char*)malloc(512);
67                  strcpy(accidente[nr_total_accidente].locatie,
                        x);
68                  nr_total_accidente++;
69              }
70
71
72          }
73          char*total_accidente = (char*)malloc(2048);
74          total_accidente[0] = '\0';
75          while(nr_accidente_din_thread <
                nr_total_accidente){//la accidente trimit la
                toti clientii indiferent de locatia lor
76              strcat(total_accidente, "Accident:");
77              strcat(total_accidente,
                    accidente[nr_accidente_din_thread].locatie);
78              strcat(total_accidente, "&&");
79              nr_accidente_din_thread++;
80          }
81          printf("[thread \%d] nr accidente din thread =
                \%d, nr total = \%d\n", thread.thread_id,
                nr_accidente_din_thread, nr_total_accidente);
82          printf("[thread \%d] scriu catre client
                >\%s<\n", thread.thread_id, total_accidente);
83          if(write(broadcast_descriptor, total_accidente,
                strlen(total_accidente)) < 0){
84              perror("Eroare la write");
```

```
85              quit = 1;
86              break;
87          }
88
89          //astepta locatia trimisa de client
90          poll_code = poll(location_fd, 1, 1000);
91          if(poll_code == 1){//clientul a trimis locatia
                sau un cod de succes, daca a ajuns la
                destinatie
92              memset(msg, 0, 100);
93              int codr = read(location_fd[0].fd, msg, 100);
94
95              if(codr <= 0){
96                  if(errno != EWOULDBLOCK){
97                      perror("Eroare la citire din
                            socket");
98                      break;
99                  }
100             }
101             if(strcmp(msg, "Succes") == 0){//clientul a
                    ajuns la destinatie
102                 quit = 1;
103             }else{
104                 char* aux = (char*)malloc(100);
105                 memset(aux, 0, 100);
106                 strcpy(aux, msg);
107                 char * x = (char*)malloc(100);
108                 strcpy(x, strtok(aux, "&&"));
109                 client_speed = atoi(x);
110                 strtok(NULL, "&&");
111                 char*street = strtok(NULL, "&&");
112                 char*number = strtok(NULL, "&&");
113                 int i = 0;
114                 while(number[i] <= '9' && number[i] >=
                        '0' && i <= strlen(number))i++;
115                 number[i] = '\0';
116
117                 int street_number = atoi(number);
118                 printf("[thread \%d] viteza client =
                        \%d, locatie client = \%s \%d\n",
                        thread.thread_id, client_speed,
                        street, street_number);
119                 char* events = get_events(street,
                        street_number, event_sent,
                        options);//TODO adauga parametru si
```

```
                    options(alea alese de utilizator la
                        inregistrare)
120                 printf("[thread \%d] scriu catre client
                        \%s\n", thread.thread_id, events);
121                 if(write(broadcast_descriptor, events,
                        strlen(events) + 1) < 0){
122                     perror("Eroare la write");
123                     quit = 1;
124                     break;
125                 }
126             }
127         }
128     }
129
130
131
132     printf("[thread \%d] Inchidere thread\n",
            thread.thread_id);;
133     total_threads--;
134     close(broadcast_descriptor);
135     close(sd);
136     close((uintptr_t)arg);
137     return NULL;
138 }
```

5 Concluzii

Acest proiect poate fi imbunatatit prin implementarea unei interfete grafice care sa afiseze o harta actualizata in timp real. De asemenea, daca s-ar putea modifica optiunile alese de clienti si dupa ce s-au inregistrat, utilizatorii ar avea parte de o experienta mai buna a aplicatiei.

In loc sa folosesc 2 socketi, as putea sa folosesc unul singur daca as adauga noi reguli in comunicare client-server astfel incat sa trimit la un interval de n secunde toate mesajele(accident, eveniment, etc). Acest lucru se poate realiza, daca as pune intr-o coada mesajele care se aduna in acel interval de timp.

Pentru a imbunatati viteza serverului, as putea sa apelez functia accept() in threaduri, adica sa fac un threadpoll, sau as putea sa implementez un server TCP prethreaded.

6 Bibliografie

References

1. https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_71/rzab6/poll.htm
2. https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c
3. https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/cliTcpNr.c
4. https://profs.info.uaic.ro/ computernetworks/files/7rc_ProgramareaInReteaIII_Ro.pdf
5. https://www.qt.io/
6. https://www.youtube.com/playlist?list=PL2D1942A4688E9D63
7. https://profs.info.uaic.ro/ computernetworks/cursullaboratorul.php
8. https://doc.qt.io/