Monitorizarea Traficului

Brezuleanu Mihai Alexandru, IIA4

Facultatea de Informatică Iași

1 Introducere

Monitorizarea traficului constă într-un sistem capabil să gestioneze traficul auto și să ofere informații virtuale șoferilor, în timp real. Astfel sistemul oferă date actuale despre evenimentele de pe drum, permițând clienților(șoferii angrenați în trafic) să vizualizeze densitatea traficului, diverse obstacole sau accidente rutiere, viteza maximă admisă/recomandată pe o anumită porțiune de drum precum și alte modificări din condițiile de drum.
Mai mult decât atât, sistemul permite doar utilizatorilor înscriși pentru aceste servicii să primeasacă informații despre vreme, evenimente sportive sau prețuri pentru combustibili la stațiile peco aflate în vecinătate.
Sistemul primește la o anumită frecvență de timp update-uri referitoare la locația și viteza cu care șoferii conectați circulă la momentul respectiv, și ținând cont de evenimentele curente, actualizează în mod automat informațiile, transmițând înapoi utilizatorilor noile condiții de drum.

2 Tehnologiile utilizate

Aplicaţia va fi implementată în limbajul C, în timp ce pentru comunicarea în reţea va fi utilizat protocolul TCP/IP întrucât este un protocol de transport orientat conexiune, fără pierdere de informații. Astfel se garantează că mesajele transmise dinspre server spre clienți, respectiv dinspre clienți către server, nu sunt eronate, respectă ordinea în care au fost trimise şi mai mult, în cazul în care acestea sunt pierdute, vor fi retransmise. Pentru programarea în reţea va fi folosit API-ul Socket-BSD (Berkeley System Distribution).

Pentru ca clienții să poată fi tratați concurent, se va utiliza un server TCP concurent care deservește clienții prin crearea unui thread dedicat tratării fiecărui client/conexiune. Mai exact, în cadrul fiecărui thread creat serverul realizează comunicarea/transferul de date către și dinspre clientul pe care îl tratează.

Pentru facilitarea lucrului cu date se vor utiliza baze de date, folosindu-se în acest sens API-ul SQLite în timp ce pentru crearea unei interfețe vizuale se va folosi header-ul graphics.h ce oferă acces la o librărie grafică menită să ușureze reprezentarea vizuală a informațiilor.

3 Arhitectura aplicației

3.1 Conceptele implicate

Așa cum a fost menționat anterior se va folosi un server TCP concurent care deservește clienții prin crearea unui thread pentru fiecare conexiune. Mai exact, vor fi implementate două aplicaţii: una de tip client(care va primi, într-un format stabilit, comenzile de la tastatură), şi una de tip server(care va primi comenzile de la clienți, le va procesa, și va returna clienților răspunsul la comanda respectivă).

Aplicația server va crea așadar câte un thread pentru fiecare client care vine să se conecteze la server, în timp ce aplicaţia client va avea în afară de procesul "părinte", două thread-uri: unul dintre acestea va prelua comenzile și le va trimite la server(deci va fi cel cu care va interacţiona utilizatorul), iar cel de-al doilea va recepționa răspunsurile date de server și le va afișa pe ecran.

Observație: În ceea ce privește informațiile despre vreme, articole sportive, prețurile carburanților, numele străzilor, viteza maximă admisă pe o anumită stradă, etc. acestea vor fi reținute în baza de date.

3.2 Diagrama aplicației detaliată

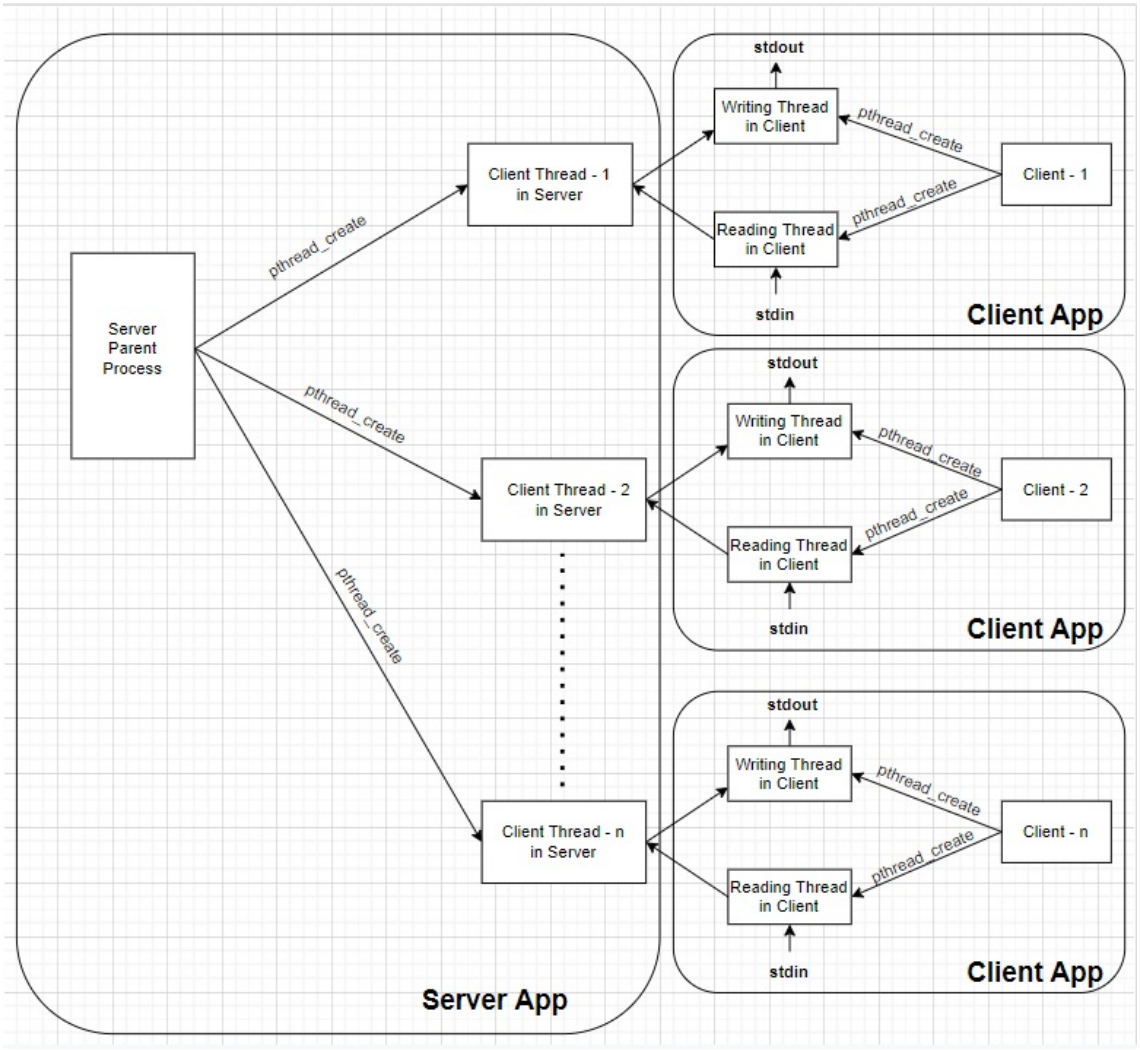

Fig. 1. Diagrama aplicațiilor Server și Client

4 Detalii de implementare

Pe partea de client, thread-ul care se ocupă de citirea comenzilor de la tastatură va verifica ca, șirul de caractere introdus de la tastatură să corespundă unei comenzi valide (să respecte un format standard), iar în caz afirmativ acesta va fi trimis prin intermediul socket-ului specific clientului către server.

Tot pe partea de client, thread-ul care va recepta răspunsul dat de server(cel de-al doilea thread al aplicației), primește tot prin intermediul socket-ului specific clientului, răspunsul aferent comenzii anterior introduse și afișează după caz, răspunsul în consola clientului.

În cadrul aplicației Server, fiecare thread corespunzător clientului pe care îl tratează, va primi prin intermediul socket-ului, comanda transmisă de acesta, și o va prelucra, folosindu-se de informațiile din baza de date. Astfel, după ce toate modificările au fost efectuate în cadrul aplicației Server, aceasta trimite înapoi copilului răspunsul așteptat.

Observații:

1. Pentru un client se va crea o structură menită să rețină informații despre acesta(descriptorul socket-ului returnat de accept(), un id unic, viteza, poziția, etc.).
   Mai mult decât atât, pentru a se putea ține o evidentă a clienților conectați într-un anume moment la server, se va utiliza o "coadă" reprezentată printr-un vector de pointeri ce vor conține adresele unor zone de memorie în care se află variabile de tipul structului prezentat anterior(un vector de pointeri ce "pointează" la o variabilă de tip struct).
2. Un client va fi adăugat în "coadă" atunci când se va stabili conexiunea cu el, și va fi scos în momentul în care se deconectează de la server.
3. Utilizatorii vor interacționa cu aplicația Client prin intermediul unei console cu comenzi.

4.1 Secvențe de cod explicate

```
// crearea unui socket
if ((sd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
  {
    perror ("[server]Eroare la socket().\n");
    return errno;
  }
// utilizarea optiunii SO_REUSEADDR ce permite repornirea imediata a serverului dupa inchidere
int on=1;
setsockopt(sd,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on));

// pregatirea structurilor de date
bzero (&server, sizeof (server));
bzero (&from, sizeof (from));


// stabilirea familiei de socket-uri
  server.sin_family = AF_INET;
// acceptam orice adresa
  server.sin_addr.s_addr = htonl (INADDR_ANY);
// utilizam un port utilizator
  server.sin_port = htons (PORT);

// atasam socketul
if (bind (sd, (struct sockaddr *) &server, sizeof (struct sockaddr)) == -1)
  {
    perror ("[server]Eroare la bind().\n");
    return errno;
  }

// punem serverul sa asculte daca vin clienti sa se conecteze
if (listen (sd, 10) == -1)
  {
    perror ("[server]Eroare la listen().\n");
    return errno;
  }
```

Fig. 2. Crearea serverului TCP în aplicația Server

```c
//Structura pentru client
typedef struct{
    struct sockaddr_in address;
    int sockfd;
    int uid;
    char name[32];
    int info;//0-nu e abonat la stiri ; 1-altfel
    int nrstr;//numarul strazii
    int speed;//viteza
    int borna;//unde pe strada se afla
    int xpoz,ypoz;//pozitia in mapa
} client_t;

client_t *clients[MAX_CLIENTS];
```

Fig. 3. Structura destinată unui client

```c
   while(1){
       socklen_t clilen = sizeof(cli_addr);
       connfd = accept(listenfd, (struct sockaddr*)&cli_addr, &clilen);

       //Verific daca a fost atins numarul maxim de clienti
       if((cli_count + 1) == MAX_CLIENTS){
           printf("Numarul maxim de clienti a fost atins. Respins: ");
           printf(":%d\n", cli_addr.sin_port);
           close(connfd);
           continue;
       }

       /* Completez campurile structurii client*/
       client_t *cli = (client_t *)malloc(sizeof(client_t));
       cli->address = cli_addr;
       cli->sockfd = connfd;
       cli->uid = uid++;
       cli->info=0;//initial nu este conectat pentru a primi informatii
       cli->nrstr=0;cli->speed=0;cli->borna=0;cli->xpoz=0;cli->ypoz=0;
       //Adaug clientul in coada si creez thread-ul
       queue_add(cli);
       pthread_create(&tid, NULL, &handle_client, (void*)cli);

       sleep(1);
   }
```

Fig. 4. Tratarea concurentă a clienților care se conectează în aplicația Server

```c
pthread_t send_msg_thread;
//cream thread-ul care se va ocupa cu trimiterea comenzilor la server
if(pthread_create(&send_msg_thread, NULL, (void *) send_msg_handler, NULL) != 0){
  printf("ERROR: pthread\n");
  return EXIT_FAILURE;
}

pthread_t recv_msg_thread;
//cream thread-ul care se va ocupa cu receptarea raspunsurilor primite de la server
if(pthread_create(&recv_msg_thread, NULL, (void *) recv_msg_handler, NULL) != 0){
  printf("ERROR: pthread\n");
  return EXIT_FAILURE;
}
```

Fig. 5. Crearea celor două thread-uri în aplicația Client

4.2 Scenarii de utilizare

În aplicația Client:

1. Fiecare utilizator trimite automat la o frecventă de 1 minut informații la server cu privire la locația și viteza cu care circulă la momentul respectiv;
2. Un utilizator poate raporta serverului un accident rutier, serverul reținând în acest sens strada pe care respectivul eveniment a avut loc;
3. Un utilizator poate raporta serverului un obstacol întâlnit pe drum ce poate pune în pericol siguranța celorlalți șoferi, serverul reținând de asemenea strada pe care se află;
4. Orice utilizator poate raporta serverului locuri unde se află echipaje ale poliției(radar sau filtru);

In aplicația Server:

1. Serverul trimite către fiecare client informații despre viteza maximă admisă/recomandată cu care trebuie să circule pe porțiunea de drum în care se pozitionează;
2. Serverul trimite către toți clienții informații despre un blocaj/accident rutier sau despre prezența unui echipaj de poliție, în momentul în care la rândul său primește această informație de la un client;
3. Serverul trimite utilizatorilor(șoferilor) avertizări privind depășirea limitelor de viteză, acolo unde este cazul;
4. Serverul trimite informații despre vreme, evenimente sportive sau prețuri pentru combustibili la stațiile peco doar acelor utilizatori care au selectat aceste opțiuni;

Observații: Aplicația va reprezenta grafic poziționarea în rețeaua rutieră a utilizatorilor conectați în acel moment în sistem.

5 Concluzii

Aplicația "Monitorizarea traficului" va permite utilizatorilor(participanții la trafic) să vizualizeze rețeaua rutieră în timp real, având astfel opțiunea de a lua decizii cu privire la traseul optim în funcție de ambuteiajele și restricțiile existente. De asemenea sistemul oferă șoferilor posibilitatea de a-i informa și pe ceilalți conducători auto despre diversele evenimente întâlnite în trafic. Nu în ultimul rând, aplicația este una utilă întrucât mai poate oferi utlizatorilor care optează pentru aceste servicii, informații despre vreme, prețuri de carburanți sau chiar evenimente sportive, permițându-le acestora să fie mereu informați cu privire la ultimele noutăți.

5.1 Îmbunătățirea soluției propuse

Datorită faptului că mai multe thread-uri ce aparțin aceluiași proces pot accesa în comun date declarate în cadrul procesului apare necesitatea sincronizării acestui acces. Astfel, un mecanism pe care îl putem utiliza în acest sens este mutex-ul. O variabilă de tip mutex este "deținută" de un singur thread la un anumit moment. Astfel accesul la o anumită secțiune de cod (secțiune critică - care în principiu va conține modificări de date partajate între thread-uri) poate fi restrâns doar la un singur thread - cel ce "deține" mutex-ul. Celelalte thread-uri care doresc acces sunt blocate până la "eliberarea" mutex-ului, când unul din ele îl va prelua. Un exemplu în acest sens îl întâlnim atunci când mai mulți clienți se pot conecta în același timp la server, acesând sau modificând astfel informații din vectorul de structuri asignat utilizatorilor logați în aplicație.

References

1. https://app.diagrams.net/
2. https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c
3. http://www.springer.com/lncs
4. https://dzone.com/articles/parallel-tcpip-socket-server-with-multi-threading
5. https://www.geeksforgeeks.org/handling-multiple-clients-on-server-with-multithreading-using-socket-programming-in-c-cpp/
6. http://www.mario-konrad.ch/blog/programming/multithread/tutorial-04.html