Raport Tehnic

Matcovici Georgiana

Facultatea de Informatică, Universitatea "Alexandru Ioan Cuza" din Iași

1 Introducere

Proiectul "Sistem inteligent de monitorizare a traficului" presupune realizarea unei aplicații care să implementeze soluții pentru probleme frecvente in viața cotidiană, precum lipsa informațiilor in timp real în trafic, accidentele și blocajele rutiere. Aplicația utilizează modelul client-server și oferă urmatoarele funcționalități: conectarea simultană a mai multor mașini, raportarea unui accident de catre un participant în trafic și transmiterea informației către toți ceilalți participanți, informare cu privire la restricții de viteză pe anumite porțiuni, și, opțional, oferirea unor actualizări despre vreme, evenimente sportive sau prețuri pentru combustibili.

2 Tehnologii aplicate

În realizarea aplicației, pentru a asigura comunicarea dintre clienți și server, am folosit un socket TCP, în care server-ul utilizează un model concurent, ceea ce permite gestionarea simultană a conectării mai multor clienți, prin creearea a câte unui thread care se ocupă de fiecare client în parte. Concurența a fost implementată folosind biblioteca <pthread.h>. Motivația alegerii protocolului TCP este necesitatea ca informații importante, precum o notificare cu privire la un accident sau o atenționare referitoare la condiții meteo dificile pentru circulația rutieră să ajungă la clienți în mod sigur și fără a avea pierderi de date. Pentru gestionarea notificărilor la intervale periodice de timp (de exemplu, fiecare client trimite automat către server viteza actualizată la fiecare 60 s, serverul trimite periodic informatii optionale solicitate de catre clienti), am folosit semnale (SIGALARM). De asemenea, pentru stocarea informatiilor necesare (informatii utile cu privire la fiecare strada: anumite restrictii de viteza, restrictii pentru vehicule de mare tonaj, daca exista un accident sau blocaj pe strada respectiva, daca exista o benzinarie si informatiile despre vreme si evenimente sportive) am utilizat o baza de date-SQLite. Pentru a retine cate o lista cu acei clienti care doresc informatii suplimentare, am utilizat structuri de date din <unordered_set>.

3 Structura aplicației

Aplicația are o arhitectură de tip client-server, în care clienții solocită informații și primesc notificări de la server. Structura detaliată a aplicației este descrisă în următoarea diagramă:

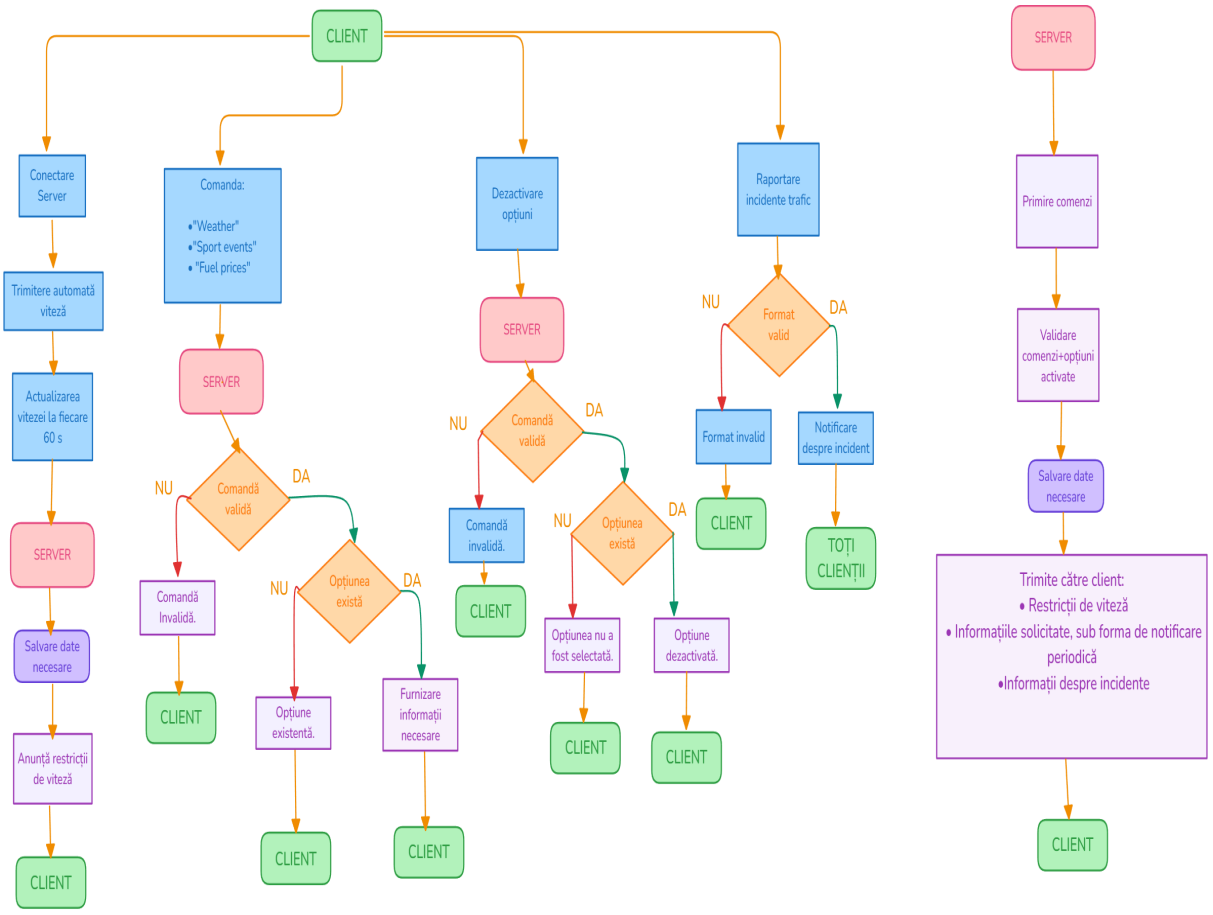

Fig. 1. Diagrama aplicației

4 Aspecte de implementare

Secțiuni de cod relevante ale proiectului sunt:

(a) Implementarea protocolului de comunicare TCP

- Server : Server-ul creează un socket pentru comunicare și ”ascultă” conexiunile clienților.

```
if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{
    perror("[server]Eroare la socket().\n");
    return errno;
}

int on = 1;
setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));

bzero(&server, sizeof(server));
bzero(&from, sizeof(from));

server.sin_family = AF_INET;

server.sin_addr.s_addr = htonl(INADDR_ANY);
server.sin_port = htons(PORT);

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

if (listen(sd, 2) == -1)
{
    perror("[server]Eroare la listen().\n");
    return errno;
}
```

Fiecare conexiune este delegată funcției treat(), care rulează intr-un thread separat, iar această funcție apelează funcția raspunde(), care gestionează interacțiunea cu clientul.

```
        pthread_create(&th[i], NULL, &treat, td);
    }
};
static void *treat(void *arg)
{
    struct thData tdL;
    tdL = *((struct thData *)arg);
    printf("Car #%d connected!\n", tdL.idThread + 1);
    fflush(stdout);
    pthread_detach(pthread_self());
    raspunde((struct thData *)arg);
    close((intptr_t)arg);
    return (NULL);
};

void raspunde(void *arg)
{
```

- Client : Aici clientul inițiază o conexiune TCP către server (folosind un port și o adresă IP specificate ca argumente).

```
port = atoi(argv[2]);

if ((sd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
{
    perror("Eroare la socket().\n");
    return errno;
}

server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr(argv[1]);
server.sin_port = htons(port);

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

Scenariu de utilizare: un client care se conectează la server trimite automat către acesta viteza cu care circulă, serverul primește mesajul și trimite înapoi către client viteza cu care este recomandat să circule, pe baza condițiilor meteo (salvate și actualizate periodic).

(b) Trimiterea automată către server a vitezei, la un anumit interval

Aici se creează un thread pentru actualizarea vitezei.

```
pthread_t SpeedUpdater;
if (pthread_create(&SpeedUpdater, NULL, UpdateSpeed, NULL) != 0)
{
    perror("Eroare creare thread.\n");
    return errno;
}
```

La un interval dat, este generat semnalul SIGALARM, la recepționarea căruia este actualizată viteza.

```
void handler_SIGALARM(int sig)
{

    int RandomSpeed;
    srand(time(0));
    RandomSpeed = rand() % 120;
    sprintf(myCommand, "Updated Speed: %d", RandomSpeed);
    if (write(sd, myCommand, sizeof(myCommand)) <= 0)
    {
        perror("[client]Eroare la write() spre server.\n");
        return errno;
    }
    alarm(5);
}

void *UpdateSpeed(void *arg)
{
    signal(SIGALRM, handler_SIGALARM);
    alarm(5);
    while (1)
    {
        pause();
    }
}
```

5 Concluzii

Posibile îmbunătățiri ale aplicației sunt: implementarea unui mecanism securizat de autentificare pentru clienți, furnizarea la cerere către un client a distanței dintre strada pe care se află și o anumită stradă.

6 Referințe bibliografice

https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php
https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html
https://man7.org/linux/man-pages/man2/alarm.2.html
https://man7.org/linux/man-pages/man0/pthread.h.0p.html
https://www.sqlite.org/cintro.html
https://cplusplus.com/reference/unordered_set/unordered_set/