Mersul Trenurilor

Ciobanu Andra Maria

andra.ciobanu@info.uaic.ro
ciobanuandramaria@gmail.com
Facultatea de Informatica Iasi
https://www.info.uaic.ro/

Abstract. Aceasta lucrare reprezinta documentatia proiectului "Mersul Trenurilor".

Keywords: C/C++ · Linux · Retele de calculatoare

1 Introducere

1.1 Viziunea generală și obiectivele proiectului

Scopul aplicatiei este de a monitoriza trenurile, putand fi folosita in gari pentru informare sau administrare, dar si pentru calatori permitand o mai buna orientare in timpul calatoriei sau in momentul asteptarii unui tren in statie. Clientii vor putea alege dintre comenzile din meniul prezentat de program in urma carora vor primi informatii relevante. Toate schimbarile produse, intarzieri sau estimari ale sosirilor in statie diferite de programul initial, vor fi comunicate acestora in timp real.

Programul trenurilor din acea zi: Aceasta comanda va asigura transmiterea programului din ziua respectiva (numarul trenului, traseul sau si orele de sosire si plecare din statii) catre fiecare client la fiecare 30 de secunde sau la cererea utilizatorului.

Plecarile din urmatoarea ora: Clientii pot apela la aceasta comanda pentru a cere informatii despre plecarile trenurilor din urmatoarea ora, dintr-o statie specificata.

Sosirile din urmatoarea ora: Clientii pot apela la aceasta comanda pentru a cere informatii despre sosirile trenurilor din urmatoarea ora, intr-o statie specificata.

Intarziere: Clientii pot inregistra o intarziere a unui tren intr-o anumita statie, specificand si numarul de minute. Comanda functioneaza si pentru cazul in care se estimeaza ca trenul ajunge mai devreme in statie. Odata inregistrata, intarzierea va fi notificata tuturor utilizatorilor conectati.

2 Tehnologii Aplicate

2.1 TCP(Transmission Control Protocol)

Protocolul implementat este TCP concurent, permitand conectarea si executarea comenzilor a mai multor clienti in acelasi timp, fara a fi nevoiti sa astepte. Oferă o comunicare sigura între dispozitivele conectate în rețea (acest lucru se realizează prin confirmarea și retransmiterea datelor în cazul în care se pierd sau se corup), fiind necesar pentru a asigura o actualizare corecta a bazei de date si transmiterea corecta a mesajelor despre trenuri intre server si clienti. Monitorizează fluxul de date pentru a asigura că receptorul poate gestiona volumul de date trimise de expeditor. Dacă receptorul nu poate procesa datele suficient de rapid, TCP va reduce temporar rata de transmitere pentru a evita pierderea de date.

2.2 Threads

Utilizarea firelor de execuție este importantă pentru gestionarea concurentă a cererilor de la clienți. Fiecare cerere poate fi tratată într-un fir de execuție separat, permițând astfel serverului să gestioneze multiple cereri simultan și să ofere răspunsuri rapide la solicitările clienților.

2.3 Sockets

În contextul implementării serverului și comunicării cu clienții, utilizarea bibliotecilor de sockets este esențială. Acestea permit crearea și gestionarea conexiunilor de rețea între server și clienți folosind protocolul TCP. Sockets facilitează trimiterea și primirea datelor între server și clienți prin intermediul rețelei.

2.4 XML Database

Pentru a procesa și actualiza datele intr-o maniera sigura, sunt folosite fișierele XML, ce stocheaza informațiile corespunzătoare despre mersul trenurilor, plecări, sosiri, întârzieri și estimări de sosire.

3 Arhitectura Aplicației

Serverul va permite conectarea concurenta a mai multor clienti si comunicarea cu acestia, cu ajutorul protocolului TCP si a implementarii socketurilor. Acesta va crea 2 threaduri separate: unul pentru trimiterea programului la fiecare 30 de secunde catre toti clientii conectati si unul pentru a procesa intarzierile inregistrate si a le trimite tuturor clientilor conectati. Dupa ce au fost initializate structurile pentru stabilirea conexiunii, fiecare client va cere conectarea si, pe masura ce sunt acceptati, serverul va crea un thread separat pentru fiecare dintre ei. Clientul va transmite prin socket comenzile catre threadul asociat lui si acesta le va executa cu ajutor unor functii, scriind rezultatul inapoi catre client prin intermediul unui socket. In client, dupa realizarea conexiunii, se va crea un thread pentru a primi informatiile de la server si un loop pentru a trimite comenzile. Bucla infinita se opreste in momentul in care clientul alege comanda quit, conexiunea cu serverul inchizandu-se. De asemenea, functiile apelate in threadurile serverului pentru a executa comenzile dorite au acces la informatiile din baza de date XML in care sunt stocate toate actualizarile despre programul trenurilor.

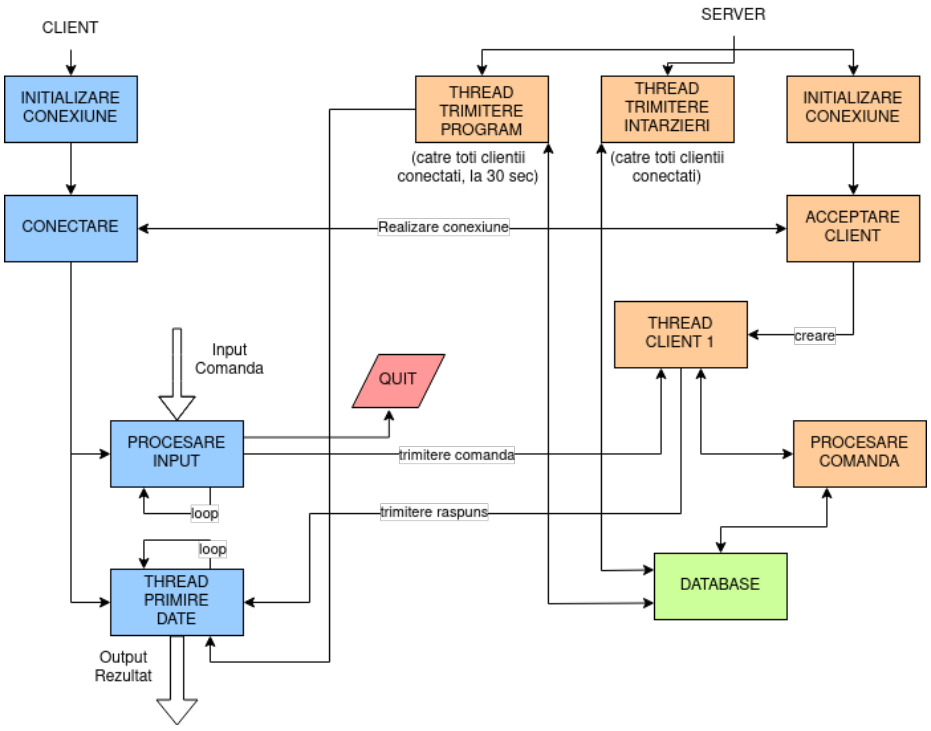

Fig. 1. Comunicarea client - server

4 Aspecte de Implementare

Protocolul utilizat este TCP concurent pentru a fi posibila conectarea mai multor clienti si procesarea comenzilor fara a astepta. Comunicarea intre server si fiecare client este realizata prin intermediul socketurilor.

4.1 Partea de Client

```
//////////////////////initializare socket
 if ((sock_descr = socket(AF_INET, SOCK_STREAM, 0)) == -1)
 {
   perror("[client]Eroare la creare socket\n");
   return errno;
 }
   server.sin_family = AF_INET;
   server.sin_addr.s_addr = inet_addr(argv[1]);
   server.sin_port = htons(port);
//////////////////////conectare
   if (connect(sock_descr, (struct sockaddr *)&server, sizeof(struct sockaddr)) == -1)
   {
     perror("[client]Eroare la conectarea la server\n");
     return errno;
   }
```

Fig. 2. Client: comunicare prin socket

Crearea unui socket: Functia socket() este utilizata pentru a crea un nou socket.

```
sock_descr = socket(AF_INET, SOCK_STREAM, 0);
```

Se incearca crearea unui socket cu familia de adrese AF_INET (IPv4), de tip SOCK_STREAM (socket de tip TCP), si un protocol specificat de 0 (valoarea implicita pentru protocolul specificat în funcția socket()). Daca apelul functiei socket() intoarce -1, se afiseaza un mesaj de eroare cu perror() si se returneaza valoarea errno care conține codul specific al erorii.

Initializarea adresei serverului: Se seteaza adresa IP si portul serverului catre care se doreste conectarea.

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

Se converteste adresa IP din formatul string la reprezentare binara utilizand functia inet_addr(). Argumentul argv[1] reprezinta adresa IP a serverului data ca argument la program. server.sin_port = htons(port): Seteaza portul serverului pentru conexiune. Functia htons() este folosita pentru a asigura conversia corecta a ordinii octetilor pentru port la ordinea de rețea.

Conectarea la server: Functia connect() este folosita pentru a stabili o conexiune la server folosind socket-ul creat anterior.Se incearca conectarea la server folosind descriptorul de socket sock_descr si adresa variabilei server (structura sockaddr_in) care contine informatiile necesare pentru conectare. Daca apelul functiei connect() intoarce -1, se afiseaza un mesaj de eroare cu perror() si se returneaza valoarea errno care contine codul specific al erorii.

```
connect(sock_descr, (struct sockaddr *)&server,
sizeof(struct sockaddr));
```

Gestionarea mesajelor cu serverul: Functia fork() creeaza un proces copil care se va ocupa de primirea mesajelor de la server folosind functia read() ce ia ca parametri descriptorul socketului, bufferul in care vom retine informatia si dimensiunea pe care o vom citi.

```
if (read(sock_descr, raspuns, sizeof(raspuns)) < 0)
    {
            perror("eroare la read() program\n");
             exit(errno);
    }
```

In procesul tata (atunci cand pid-ul este diferit de 0), preluam datele de la utilizator si trimitem comanda serverului prin functia write() avand ca parametri descriptorul socketului, bufferul in care am retinut comanda si dimensiunea pe care o vom scrie.

```
if (write(sock_descr, command, sizeof(command)) <= 0)
    {
    perror("Eroare la scrierea catre server\n");
           exit(errno);
    }
```

Fiecare proces contine o bucla: prima, cea a procesului tata, este oprita in momentul intalnirii comenzii "quit" trimitand totodata si un semnal procesului fiu pentru a-l opri.

```
if(strcmp(command,"quit")==0)
    {
          run=false;
          kill(pid,9);
    }
```

```
int pid = fork();
if (pid == -1)
{
  perror("Error in fork.\n");
  return errno;
}

if (pid)
{
  printf("\n\n////////////////////////////////////////////////////////\n\n");
  printf("              MERSUL TRENURILOR\n\n");
  printf("\n\n////////////////////////////////////////////////////////\n\n");
  fflush(stdout);
  char command[50];
  bool run = true;
  ;
  while (run == true)
  {
    printf("\n1.Status Sosiri: sosiri <statie>\n2.Status Plecari: plecari ·
    fflush(stdout);
   fgets(command, sizeof(command), stdin);
   command[strlen(command) - 1] = '\0';

   if (write(sock_descr, command, sizeof(command)) <= 0)
   {
    perror("Eroare la scrierea catre server\n");
    exit(errno);
   }
   sleep(2);

   if (strcmp(command, "quit")==0)
   {
    run = false;
    kill(pid, 9);
   }
  }
  close(sock_descr);
  exit(0);
}
else
{
  char raspuns[10000];
  while (1)
  {
   bzero(raspuns, sizeof(raspuns));

   if (read(sock_descr, raspuns, sizeof(raspuns)) < 0)

   {
    perror("eroare la read() program\n");
    exit(errno);
   }

   printf("%s", raspuns);
   fflush(stdout);
  }
}
```

Fig. 3. Client: Trimiterea si primirea mesajelor catre si de la server

4.2 Partea de Server

```
//////////////////initializare socket
   if ((sock_descr = socket(AF_INET, SOCK_STREAM, 0)) == -1)
   {
       perror("[server]Eroare la creare socket\n");
       return errno;
   }
   int on = 1;
   setsockopt(sock_descr, 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(sock_descr, (struct sockaddr *)&server, sizeof(struct sockaddr)) == -1)
   {
       perror("[server]Eroare la bind\n");
       return errno;
   }
   if (listen(sock_descr, 2) == -1)
   {
       perror("[server]Eroare la listen\n");
       return errno;
   }

   while (1)
   {
       int client;
       struct Thread *td;
       socklen_t len = sizeof(from);
       printf("Port->%d\n", PORT);
       fflush(stdout);
//////////////////conectare client
       if ((client = accept(sock_descr, (struct sockaddr *)&from, &len)) ==-1)
       {
           perror("[server]Eroare la acceptarea clientului\n");
           continue;
       }
////////crearea unui fir de executie separat pentru clientul acceptat
       td = (struct Thread*)malloc(sizeof(struct Thread));
       td->idThread = i++;
       td->client = client;
       pthread_create(&th[i], NULL, &executa, td);

   }
```

Fig. 4. Server: comunicare prin socket

Se creeaza un socket folosind `socket(AF_INET, SOCK_STREAM, 0)`. Acesta este un socket de tip TCP/IP (familia de adrese `AF_INET` si tipul `SOCK_STREAM`). Apoi, se seteaza optiunea `SO_REUSEADDR` folosind `setsockopt()`. Aceasta optiune permite reutilizarea adresei si portului, permitand serverului sa reporneasca rapid

dupa inchidere. Se initializeaza structurile `server` si `from` folosind `bzero()` pentru a le seta la zero. Se configureaza adresa si portul serverului (`server.sin_family`, `server.sin_addr.s_addr`, `server.sin_port`) pentru a asculta pe orice adresa (`INADDR_ANY`) la portul specificat (`PORT`). Apelul `bind()` este folosit pentru a lega socket-ul la adresa si portul specificate. Daca acest lucru esueaza, se afiseaza un mesaj de eroare si se returneaza valoarea `errno`.

Se foloseste `listen()` pentru a marca socket-ul sa asculte conexiuni de la clienti. Daca exista o eroare la `listen`, se afiseaza un mesaj de eroare si se returneaza valoarea `errno`. Se intra intr-un ciclu `while(1)` pentru a accepta conexiuni de la clienti folosind `accept()`. In momentul in care se accepta o conexiune de la un client, se creeaza un nou fir de executie (thread) utilizand `pthread_create()` pentru a trata cererea clientului. Se aloca memorie pentru o structura de tip `struct Thread` care contine informatii despre noul fir de executie, precum ID-ul firului si descriptorul de client acceptat.

```
void *timer(void *arg)
{
    char raspuns[10000];
    while (1)
    {
        bzero(raspuns, sizeof(raspuns));
        sleep(30);
        ProgramTrenuri(raspuns);
        for (int i = 0; i < 10; i++)
        {
            if (clients[i])
            {
                if (write(clients[i], raspuns, sizeof(raspuns)) <= 0)
                {
                    perror("[timer]Eroare la write() catre client.\n");
                }

            }
        }
    }
}

int main()
{
    pthread_t p;
    pthread_create(&p, NULL, &timer, NULL);
```

Fig. 5. Server: Thread pentru trimiterea programului la fiecare 30 sec catre toti clientii conectati

Este creat un thread la inceputul functiei main() cu functia $pthread_create$ ce se va folosi de functia `timer` pentru a putea trimite programul tuturor clientilor conectati la fiecare 30 sec. Aceasta din urma se foloseste de un array global `clients[]` in care sunt retinuti descriptorii clientilor conectati in acel moment la server. Este apelata functia de `ProgramTrenuri` pentru a scrie in buffer mesajul pe care apoi sa il transmita clientului.

5 Concluzii

Optimizari ale aplicatiei

Fiecare client isi poate crea un cont in baza de date a aplicatiei sau se poate conecta la contul sau deja existent, putand ulterior sa se delogheze. La inchiderea aplicatiei acesta va fi delogat automat. In functie de tipul contului (calator sau administrator) acesta are acces la mai multe activitati. Calatorii vor primi informatii despre traseul trenurilor si program lor din acea zi, pot vedea daca trenul in care se afla are intarziere sau daca vor ajunge mai devreme, pot vedea plecarile sau sosirile din urmatoarea ora. Administratorii pot inregistra intarzieri sau estimari ale sosirilor in statii ale unor trenuri, modificand informatiile din baza de date.

Referinte Bibliografice

1. Computer Networks - Course, https://profs.info.uaic.ro/ computernetworks/cursul-laboratorul.php
2. Computer Networks - Laboratory Guide, https://www.andreis.ro/teaching/computer-networks
3. Linux Manual, https://man7.org/linux/man-pages/index.html
4. Geeks for Geeks, https://www.geeksforgeeks.org/tcp-ip-model/?ref=lbp
5. Sisteme de Operare, https://profs.info.uaic.ro/ vidrascu/SO.html