# Sistem de Gestionare a Informatiilor Feroviare

Preda Andrei-Claudiu[2A2]

Universitatea "Alexandru Ioan Cuza" Iasi,
Facultatea de Informatica
andrei.foco@yahoo.com

**Abstract.** Acest raport prezinta implementarea unei aplicatii client-server pentru gestionarea informatiilor referitoare la trenuri folosind protocolul TCP/IP. Sunt descrise arhitectura aplicatiei, tehnologiile utilizate si detaliile de implementare.

## 1 Introducere

Scopul acestui proiect este dezvoltarea unui sistem client-server care permite gestionarea informatiilor despre plecarile, sosirile si intarzierile trenurilor. Aplicatia ofera suport pentru comunicarea intre client si server prin intermediul conexiunilor TCP. Obiectivele principale includ implementarea unui protocol personalizat si gestionarea concurenta a conexiunilor multiple prin utilizarea firelor de executie (threads).

## 2 Tehnologii Aplicate

Aplicatia utilizeaza urmatoarele tehnologii si concepte:

- **TCP/IP**: Protocolul TCP este utilizat pentru asigurarea unei comunicatii fiabile intre client si server.
- **Sockets**: Atat clientul, cat si serverul folosesc socket-uri pentru a trimite si primi date.
- **Pthreads**: Serverul foloseste fire de executie pentru a servi mai multi clienti simultan.
- **Sisteme de fisiere**: Datele despre trenuri sunt stocate si prelucrate din fisiere XML.
- **C Standard Library**: Biblioteci standard C pentru operatiuni de I/O, manipulare de siruri de caractere si gestionarea timpului.

## 3 Structura Aplicatiei

### 3.1 Concepte Folosite

In aceasta sectiune sunt descrise conceptele utilizate in implementarea serverului si clientului.

**I. Functionarea serverului multithreaded** Serverul este implementat folosind mai multe thread-uri pentru a deservi cererile clientilor. Principalele concepte includ:

- **Thread-uri:** Fiecare client este servit intr-un thread separat. Structura `Thread` gestioneaza informatiile despre fiecare thread:

```
typedef struct {
    pthread_t idThread; // ID-ul thread-ului
    int thCount;       // Nr. de conexiuni servite
} Thread;
```

- **ClientState:** Structura `ClientState` este utilizata pentru a stoca informatiile despre clienti, cum ar fi socket-ul asociat, starea de autentificare, thread-ul care serveste clientul si numele de utilizator:

```
typedef struct {
    int socket;       // Socket-ul clientului
    int logged_in;    // 1 = este autentificat, 0 = nu este
    int thread;       // ID-ul thread-ului asociat
    char username[50]; // Numele de utilizator
} ClientState;
```

- **Mutex:** Mutex-ul `mlock` este folosit pentru a sincroniza accesul la acceptarea conexiunilor:

```
pthread_mutex_t mlock = PTHREAD_MUTEX_INITIALIZER;
```

- **Functia `treat()`:** Thread-urile accepta conexiuni si gestioneaza comenzile clientului:

```
pthread_create(&threadsPool[i].idThread, NULL, &treat, (void *)(
    intptr_t)i);
```

**II. Structura datelor pentru informatii despre trenuri** Serverul utilizeaza mai multe structuri pentru a gestiona informatiile despre trenuri, exceptii si statii:

- **Structura `ExceptionInfo`:** Specifica perioada sau perioadele in care trenul circula:

```
typedef struct {
    char dataStart[11]; // Format DD.MM.YYYY
    char dataFinal[11]; // Format DD.MM.YYYY
} ExceptionInfo;
```

– **Structura `TrainInfo`:** Stocheaza informatiile detaliate despre un tren, incluzand intarzieri, statii intermediare si exceptii de circulatie:

```
typedef struct {
    char id[16];              // ID-ul trenului
    char statiePlecare[50];   // Statia de plecare
    char statieSosire[50];    // Statia de sosire
    char oraPlecare[10];      // Ora de plecare
    char oraSosire[10];       // Ora de sosire
    int intarzierePlecare;    // Intarzierea la plecare (minute)
    int intarziereSosire;     // Intarzierea la sosire (minute)
    Station statii[MAX_STATIONS]; // Lista statiilor intermediare
    int nrStatii;             // Numarul total de statii intermediare
    ExceptionInfo exceptii[MAX_EXCEPTIONS]; // Lista exceptiilor
    int nrExceptii;           // Numarul de exceptii
    int circulaAzi;           // Indicator daca trenul circula astazi
        (1 = Da, 0 = Nu)
} TrainInfo;
```

– **Structura `Station`:** Reprezinta o statie intermediara, incluzand orele de sosire, plecare si eventuale intarzieri:

```
typedef struct {
    char nume[50];      // Numele statiei
    char oraSosire[10]; // Ora de sosire in statie
    char oraPlecare[10]; // Ora de plecare din statie
    int intarziere;     // Intarzierea in statie (minute)
} Station;
```

**III. Interactiunea client-server** Interactiunea client-server utilizeaza socketuri TCP pentru comunicarea bidirectionala. Serverul raspunde cererilor clientului in functie de comanda primita. Exista doua tipuri principale de raspunsuri, care influenteaza comportamentul clientului: 1. Raspuns simplu (mesaj scurt). 2. Raspuns multiplu (mesaj lung). Acestea sunt gestionate diferit pe baza protocolului de comunicare.

*Etapele principale ale comunicarii:*

a) **Initializarea conexiunii:**
   – Clientul creeaza un socket utilizand `socket()` si stabileste conexiunea la server cu `connect()`.
   – Serverul asculta conexiunile folosind `bind()`, `listen()` si accepta conexiunile clientilor folosind `accept()`.

b) **Trimiterea unei comenzi de catre client:** Clientul introduce o comanda in linia de comanda, care este citita cu `read()` si transmisa catre server cu `write()`:

```
printf("Introduceti comanda: ");
fflush(stdout);
read(0, comanda, sizeof(comanda));
write(socket_client, comanda, strlen(comanda) + 1);
```

c) **Procesarea raspunsului serverului:** Serverul raspunde pe baza comenzii primite. Clientul distinge intre cele doua tipuri de raspunsuri:
- **Raspuns simplu** (ex. un mesaj scurt):
  - Daca raspunsul serverului nu contine cuvantul cheie `sirLung`, atunci clientul il afiseaza direct:

```
if (!strstr(raspuns, "sirLung")) {
    printf("Raspuns de la server: %s\n\n", raspuns);
}
```

- **Raspuns multiplu** (ex. mesaj lung transmis in bucati):
  - Daca raspunsul serverului contine cuvantul cheie `sirLung`, clientul intra intr-un ciclu de citire continua pana cand primeste mesajul `END`.

```
if (strstr(raspuns, "sirLung")) {
    while (strcmp(raspuns, "END") != 0) {
        memset(raspuns, 0, sizeof(raspuns));
        read(socket_client, raspuns, sizeof(raspuns));
        if (strcmp(raspuns, "END") == 0) break;
        printf("%s\n", raspuns);
        fflush(stdout);
    }
    printf("\n");
}
```

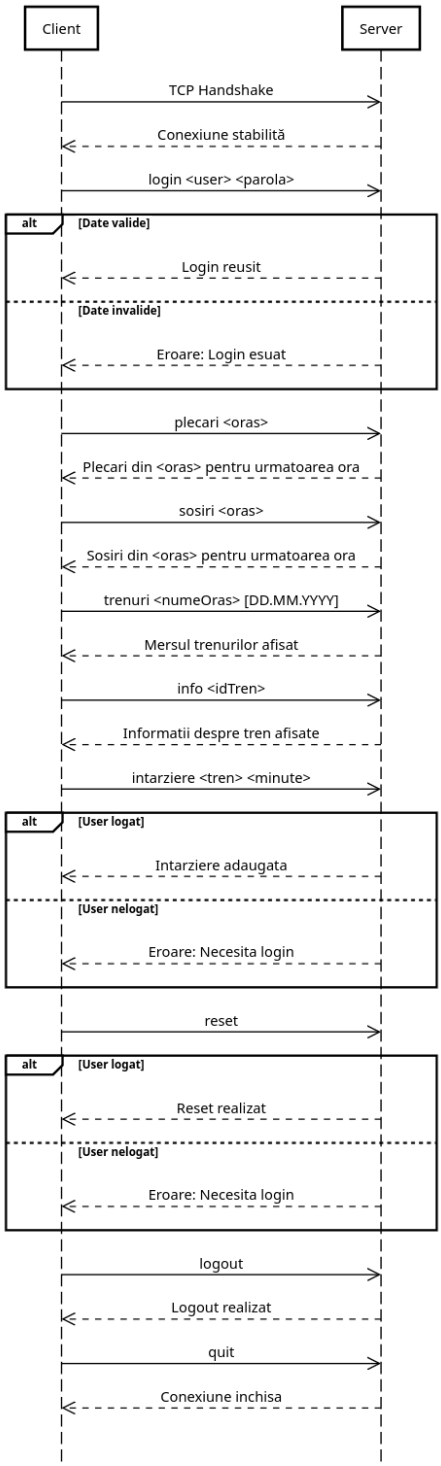
d) **Finalizarea comunicarii:**
- Daca clientul trimite comanda `quit`, serverul inchide conexiunea. Clientul gestioneaza aceasta situatie iesind din bucla principala de executie.
- Conexiunea este inchisa utilizand functia `close()`:

```
if (strcmp(comanda, "quit") == 0) {
    break;
}
close(socket_client);
```

*Avantajele acestei abordari:*

- Flexibilitate: Permite procesarea unor raspunsuri complexe, inclusiv a mesajelor lungi, fara a limita dimensiunea initiala a bufferului.
- Gestionarea eficienta a resurselor: Serverul poate trimite raspunsuri in bucati pentru a evita supraincarcarea memoriei.

### 3.2 Diagrama Aplicatiei



# 4 Aspecte de Implementare

## 4.1 Codul Serverului

Serverul este implementat in limbajul C si foloseste socket-uri si fire de executie pentru a gestiona comunicarea cu clientii.

Listing 1.1: Fragment de cod din server.c

```c
void response_manage(int cl, int idThread)
{
    char comanda[1024];
    char raspuns[1024];
    ssize_t bytes_read;

    while(1)
    {
        memset(comanda, 0, sizeof(comanda));
        bytes_read = read(cl, comanda, sizeof(comanda) - 1);
        if (bytes_read <= 0) return;

        if (strncmp(comanda, "login", 5) == 0) {
            char user[50], parola[50];
            if (sscanf(comanda, "login %49s %49s", user, parola) == 2) {
                snprintf(raspuns, sizeof(raspuns), "login reusit: %s", user
                    );
            } else {
                strcpy(raspuns, "Format invalid. Folositi: login <user> <
                    parola>");
            }
        }
        write(cl, raspuns, strlen(raspuns));
    }
}
```

## 4.2 Protocolul Aplicatiei

Protocolul defineste urmatoarele comenzi:

- **login** ⟨**user**⟩⟨**parola**⟩ - autentificare client (optional - doar pentru utilizarea comenzii **intarziere**).
- **logout** - delogheaza clientul.
- **plecari** ⟨**numeOras**⟩ [data] - afiseaza plecarile dintr-un oras (daca nu este precizata o data, se utilizeaza data curenta).
- **sosiri** ⟨**numeOras**⟩ [data] - afiseaza sosirile intr-un oras (daca nu este precizata o data, se utilizeaza data curenta).
- **trenuri** ⟨**numeOras**⟩[**DD.MM.YYYY**] - afiseaza mersul trenurilor dintr-un oras la data oferita (optional).
- **info** ⟨**idTren**⟩ - ofera informatii despre trenul specificat.

- **intarziere** ⟨**idTren**⟩⟨**statie**⟩⟨**nrMinute**⟩ - adauga o intarziere in minute pentru trenul specificat.
- **reset** - reseteaza toate intarzierile trenurilor (disponibila doar pentru clientii logati).
- **quit** - deconecteaza clientul.

### 4.3 Scenarii Reale de Utilizare

- Un client trimite comanda 'plecari Bucuresti Nord 10.06.2024' pentru a obtine plecarile din Bucuresti la o anumita data.
- Un client logat trimite comanda 'intarziere IR1660 Iasi 15' pentru a adauga o intarziere de 15 minute trenului cu ID-ul IR1660, incepand cu statia Iasi.
- Un client logat trimite comanda 'reset' pentru a reseta intarzierile trenurilor.

## 5 Concluzii

Aplicatia dezvoltata permite gestionarea eficienta a informatiilor despre trenuri prin intermediul unui protocol simplu implementat folosind TCP. In urma testarii si evaluarii functionalitatii, pot fi aduse urmatoarele imbunatatiri:

- **Adaugarea unui mecanism de autentificare mai complex** – Implementarea unor protocoale de securitate avansate, precum autentificarea pe baza de token-uri.
- **Realizarea unei interfete grafice pentru client** – O interfata grafica ar imbunatati experienta utilizatorilor, oferindu-le un mod mai intuitiv si mai prietenos de interactionare cu aplicatia.
- **Implementarea unui sistem de notificari** – Adaugarea unui sistem de notificari care sa informeze utilizatorii despre intarzierea trenurilor sau modificari de program ar imbunatati serviciile oferite.
- **Extinderea functionalitatii cu optiuni suplimentare** – Posibilitatea de a consulta informatii suplimentare, cum ar fi tarifele biletelor, disponibilitatea locurilor sau conditiile meteo in statiile respective, ar adauga valoare aplicatiei.

## Referinte

## References

1. Douglas E. Comer, *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, Prentice Hall, 2006.
2. Richard Stevens, *Unix Network Programming*, Addison-Wesley, 1998.