Mersul Trenurilor
Proiect (A) - Retele de Calculatoare

Maximiuc Teodora - B3

Ianuarie 2024

1 Introducere

Proiectul consta intr-o aplicatie de tip server/client cu scopul principal de furnizare a datelor relevante privind mersul trenurilor, precum: statusul plecărilor, statusul sosirilor, întârzierile și estimarea sosirii, informații actualizate în timp real pentru toți clienții înregistrați. Serverul va citi datele din fișierul XML și va actualiza informațiile (întârzieri și estimări de sosire) la cererea clienților. Toată logica va fi implementată în server, iar clientul va avea rolul de a solicita informații despre plecări și sosiri, cât și de a notifica eventuale întârzieri.

2 Tehnologii utilizate

2.1 Multithreading TCP/IP Server & Socket-uri

Transmission Control Protocol (TCP) este un protocol folosit de obicei de aplicații care au nevoie de confirmare a primirii datelor. Efectuează o conectare virtuală full duplex între două puncte terminale, fiecare punct fiind definit de către o adresă IP și de către un port TCP.

Pentru acest proiect am ales să folosesc un server TCP concurent, deoarece, în cadrul aplicației, prioritar este integritatea și transmiterea în siguranță a datelor. Am integrat thread-uri în proiect pentru a gestiona eficient concurența în server. Acestea permit serverului să proceseze mai multe cereri simultan, îmbunătățind astfel performanța și reducând timpul de răspuns către clienți.

Comunicarea între server și client este realizată folosind socket-uri. Socket-urile sunt utile atât pentru aplicații stand-alone, cât și pentru cele de rețea, pentru că acestea permit schimbul de informații între procese pe aceeași mașină sau prin intermediul unei rețele și permit accesul ușor la date.

2.2 Stocarea datelor: SQLite & Fișier XML

Informațiile despre utilizatori sunt salvate într-o bază de date SQL. Aceasta conține detalii despre ID-ul utilizatorului, username-ul, parola și status-ul, care ține evidența persoanelor active curent.

	username	password	Status
	Filter	Filter	Filter
1	Teo	-792095615	0
2	calator	-1411147296	0
3	user	97006	0
4	user1	97006	0
5	cala2tor	1586556	0

Figure 1: Tabela TUsers din baza de date utilizatori.db

Pentru a descrie programul trenurilor, m-am folosit de un fișier XML. Acesta conține un element principal <TrainNumber>care include subelemente denumite dupa numarul trenului. Acesta conține subelemente pentru a specifica diferite aspecte ale călătoriei:

- <Stations>: ce contine subelemente enumerate in ordinea traseului care il parcurge trenul, denumite dupa numele statiei.
- <"Nume de statie" >: subelement al elementului "Stations" care contine ora la care trenul ajunge in statia respectiva.
- <Delay>: Informații despre eventuale întârzieri (număr pozitiv dacă este întârziere, număr negativ dacă trenul vine mai devreme).
- <Status>: Valoare binara ce arata daca trenul este activ sau nu (**0** - trenul nu este in mers, **1** - este in mers)

De exemplu, un fragment al fișierului XML arată astfel:

```
<Train_Number>
    <IR1833>
        <Stations>
            <Iasi>6:00</Iasi>
            <Suceava>8:02</Suceava>
            <Cluj>15:10</Cluj>
            <Timisoara>22:58</Timisoara>
        </Stations>
        <Delay>0</Delay>
        <Status>1</Status>
    </IR1833>
</Train_Number}
```

3 Structura aplicatiei

Serverul este creat pentru a gestiona conexiunile client-server. După configurarea socket-ului și deschiderea bazei de date, serverul ascultă cereri de conectare de la clienți. Aceste conexiuni sunt gestionate în mod concurent folosind fire de execuție (sau thread-uri).

Atunci când un client solicită o conexiune, serverul acceptă cererea și inițiază un nou thread, responsabil cu citirea și interpretarea comenzilor primite de la client, efectuarea operațiilor corespunzătoare în baza de date și transmiterea rezultatelor înapoi la client.

Astfel, serverul funcționează într-un mod persistent, așteptând și tratând conexiuni de la clienți cu comunicarea în timp real.

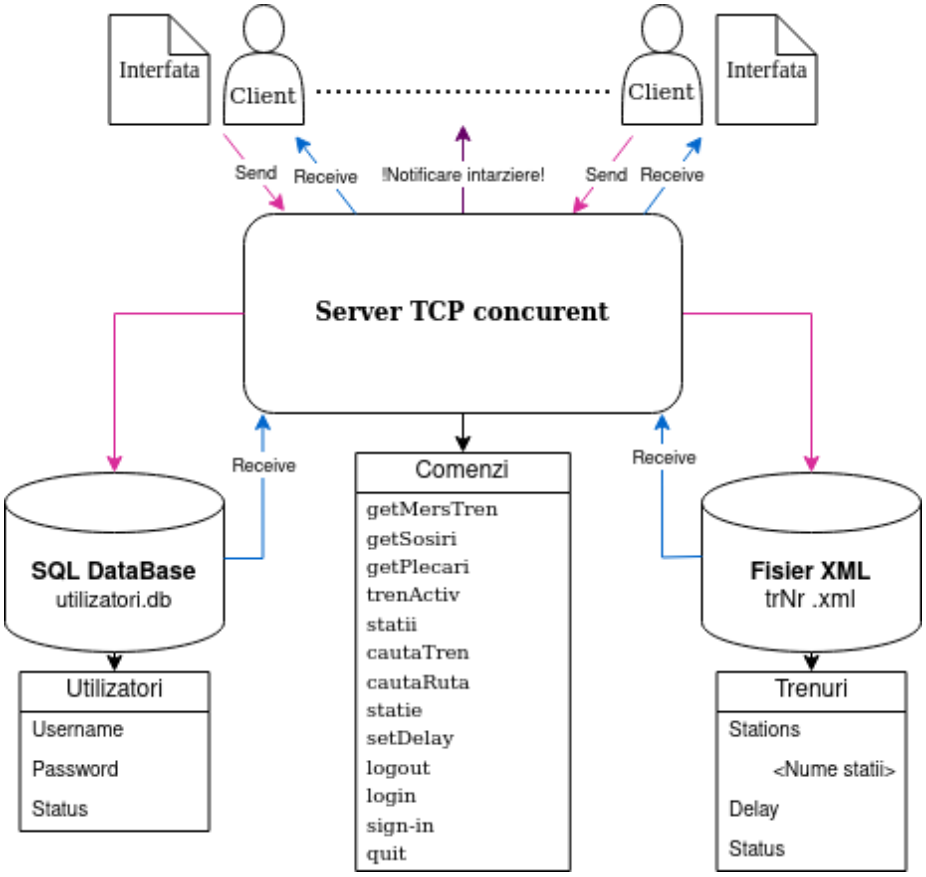

Figure 2: Diagrama aplicatiei

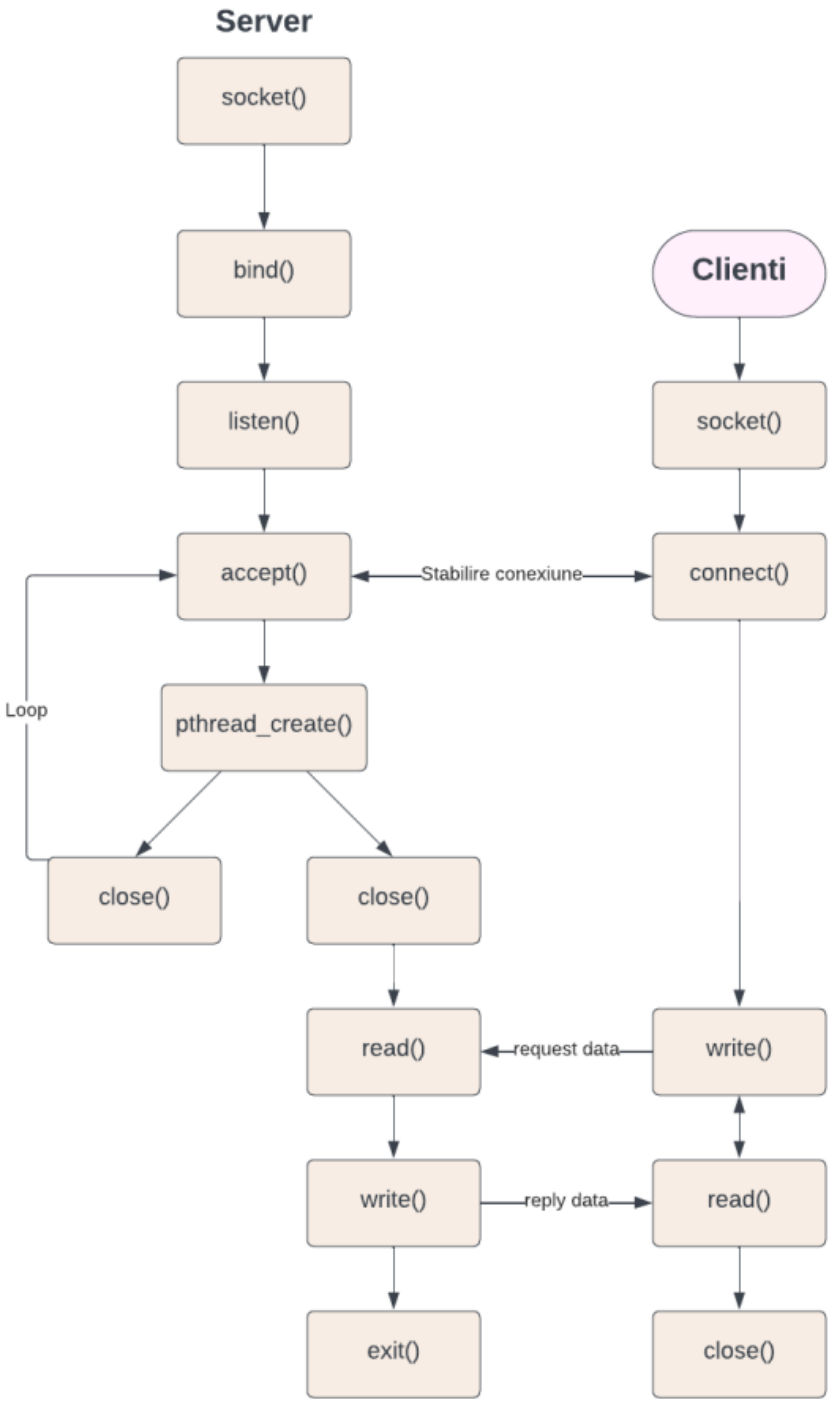
Server
socket()
bind()
listen()
accept()
Loop
pthread_create()
close()
close()
Clienti
socket()
Stabilire conexiune
connect()
read()
request data
write()
write()
reply data
read()
exit()
close()

Figure 3:

4 Aspecte de implementare

4.1 Cod relevant proiectului

Înainte ca clienții să poată solicita informații sau să raporteze întârzieri despre trenuri, aceștia trebuie să se autentifice in meniul interfetei, unde trebuie să introducă numele de utilizator și parola. Dacă informațiile acestora sunt regăsite în baza de date SQLite, care conține toți utilizatorii valizi, atunci clientul este acum autentificat, statusul său devine 1 și are acces la mersul trenurilor.

```
int authentication(sqlite3 *db, const char *username, int hash){
    sqlite3_stmt *stmt;
    char select[101];
    snprintf(select, sizeof(select), "SELECT * FROM TUsers WHERE Username='%s' AND Password=%d AND
Status=0", username, hash);

    if(sqlite3_prepare_v2(db, select, -1, &stmt, NULL) != SQLITE_OK){
        fprintf(stderr, "Error preparing SQL stmt: %s\n", sqlite3_errmsg(db));
        return 0;
    }
    int result = 0;
    if (sqlite3_step(stmt) == SQLITE_ROW)
        result = 1;
    sqlite3_finalize(stmt);
    return result;
}
```

Figure 4: Verificarea in baza de date a utilizatorilor

```
  while (1) {
    int client;
    thData *td;
    int length = sizeof(from);
    printf ("[server]Asteptam la portul %d...\n",PORT);
    fflush (stdout);
    if ( (client = accept (sd, (struct sockaddr *) &from, &length)) < 0) {
        perror ("[server]Eroare la accept().\n");
        continue;
    }
    td=(struct thData*)malloc(sizeof(struct thData));
    td->idThread=i++;
    td->cl=client;
    td->db = db;
    strcpy(td->username, "");
    strcpy(td->statie, "");
    addClient(td);

    pthread_create(&th[i], NULL, &treat, td);
  }
```

Figure 5: Conectiunea server-client si stocarea datelor clientului in struct

```c
unsigned int hashpswrd(const char *str) {
    unsigned int hash = 0;

    while (*str) {
        hash = (hash * 31) + *str++;
    }

    return hash;
}
```

Figure 6: Hashing la parola

Odata cu conectarea unui client la server, datele lui sunt stocate intr-o variabila de tip thData. Mai apoi este adaugat pe o pozitie intr-un vector de tip thData, lucru ce ajuta ulterior la trimiterea de notificarii catre clientii relevanti in legatura cu eventuale intarzieri.

```c
typedef struct thData{
    int idThread; //id-ul thread-ului tinut in evidenta de acest program
    int cl; //descriptorul intors de accept
    sqlite3 *db;
    char username[50];
    char statie[50];
}thData;
typedef struct trSort{
    char traseu[1024];
    int time;
}trSort;

thData *activeClients[MAX_CLIENTS];
trSort *sortTrains[13];

void addClient(thData *td) {
    pthread_mutex_lock(&clients_mutex);
    for (int i = 0; i < MAX_CLIENTS; ++i) {
        if (activeClients[i] == NULL) {
            activeClients[i] = td;
            break;
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}
void removeClient(thData *td) {
    pthread_mutex_lock(&clients_mutex);
    for (int i = 0; i < MAX_CLIENTS; ++i) {
        if (activeClients[i] == td) {
            activeClients[i] = NULL;
            break;
        }
    }
    pthread_mutex_unlock(&clients_mutex);
}
```

Figure 7: addClient & removeClient

La executarea comenzii `setDelay`, după setarea întârzierii, cu ajutorul vectorului ce conține clienții activi și informațiile despre aceștia, trimitem o notificare către toți clienții ce sunt în stațiile prin care trenul întârziat o să treacă.

```
    else if(strncmp(buf, "setDelay", 8) == 0) {
        char dValue[10], name_Stations[101];
        strcpy(name_Stations, "");
        sscanf(buf, "setDelay %s %s", trainNr, dValue);
        for (xmlNodePtr trains = xmlFirstElementChild(root); trains != NULL; trains =
xmlNextElementSibling(trains)) {
            if (xmlStrEqual(trains->name, BAD_CAST trainNr)) {
                xmlNodePtr stations = xmlFirstElementChild(trains);
                for (xmlNodePtr station = stations->children; station != NULL; station =
xmlNextElementSibling(station)) {
                    strcat(name_Stations, (char *)station->name);
                    strcat(name_Stations, " ");
                }
                xmlNodePtr delay = xmlNextElementSibling(stations);
                xmlNodePtr status = xmlNextElementSibling(delay);
                xmlChar *content = xmlNodeGetContent(status);
                if(strcmp(content, "0") == 0) {
                    strcpy(buf, "Trenul nu este in mers."); break;
                }
                xmlFree(content);
                if (delay != NULL) {
                    xmlNodeSetContent(delay, BAD_CAST dValue);
                    xmlSaveFormatFile("trNr.xml", doc, 1);

                    pthread_mutex_lock(&clients_mutex);
                    for (int i = 0; i < MAX_CLIENTS; ++i) {
                        if (activeClients[i] != NULL && strstr(name_Stations, activeClients[i]-
>statie)) {
                            char notif[1024];
                            if(atoi(dValue)==1) { snprintf(notif, 1024, "Atentie! Trenul %s are
intarziere de un minut.", trainNr); }
                            else if(atoi(dValue)>1) { snprintf(notif, 1024, "Atentie! Trenul %s are
intarziere de %s minute",trainNr, dValue); }
                            else{ int nr=atoi(dValue)*(-1); snprintf(notif, 1024, "Atentie! Trenul %s
vine mai devreme cu %d minute", trainNr, nr); }
                            if (write(activeClients[i]->cl, notif, 1024) <= 0) {
                                fprintf(stderr, "Error notifying client: %s\n", strerror(errno));
                            }
                        }
                    }
                    pthread_mutex_unlock(&clients_mutex);
                    break;
                }
            }
        }
    }
```

Figure 8: Comanda ”setDelay” si trimiterea notificarii

```
    else if(strncmp(buf, "login", 5) == 0) {
        sscanf(buf, "login %s %s", username, password);
        hash = hashpswrd(password);
        if (authentication(td->db, username, hash))
        {
            snprintf(update, sizeof(update), "UPDATE TUsers SET Status=1 WHERE Username='%s'",
username);
            if (sqlite3_exec(td->db, update, NULL, NULL, NULL) != SQLITE_OK)
            {
                fprintf(stderr,"Error updating user status: %s\n", sqlite3_errmsg(td->db));
            }
            strcpy(td->username, username);
            strcpy(buf, "Logat.");
         }
         else
            strcpy(buf, "Eroare la logare.");
    }
    else if(strcmp(buf, "quit") == 0 || strcmp(buf, "logout") == 0) {
        snprintf(update, sizeof(update), "UPDATE TUsers SET Status=0 WHERE Username='%s'", td-
>username);
        if (sqlite3_exec(td->db, update, NULL, NULL, NULL) != SQLITE_OK)
        {
            fprintf(stderr, "Error updating user status: %s\n", sqlite3_errmsg(td->db));
        }
        strcpy(td->username, "");
        strcpy(td->statie, "");
        if(strcmp(buf, "quit") == 0)
            {strcpy(buf, "Quitting..."); removeClient(td);}
        else
            strcpy(buf, "Deconectat.");
    }
```

Figure 9: Comenzile login, logout, quit.

Figure 10: Meniul aplicatiei realizat cu libraria GTK.

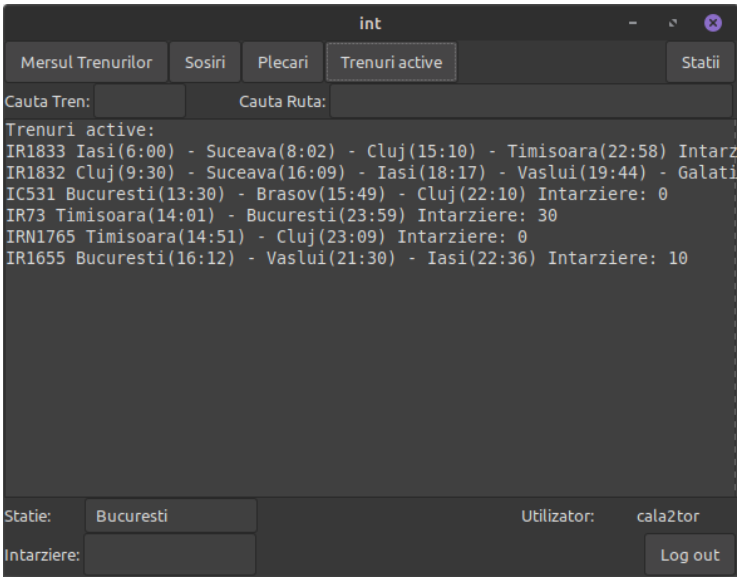

Figure 11: Interfata aplicatiei realizat cu libraria GTK.

```
void *receiveMessages(void *arg) {
    while (1) {
        if (read(sd, buf, sizeof(buf)) < 0) {
            perror("[client]Eroare la read() de la server.\n");
            break;
        }
        if (gtk_widget_get_visible(app)){
            if (strncmp(buf, "Atentie!", 8) == 0) {
                GtkTextBuffer *buffer = gtk_text_view_get_buffer(printTextView);
                GtkTextIter start, end;
                gtk_text_buffer_get_start_iter(buffer, &start);
                gtk_text_buffer_get_end_iter(buffer, &end);
                char current_text[1024];
                gtk_text_buffer_get_text(buffer, &start, &end, FALSE);
                snprintf(current_text, sizeof(current_text), "%s", gtk_text_buffer_get_text(buffer,
&start, &end, FALSE));
                char notif[4096];
                snprintf(notif, sizeof(notif), "%s\n\n%s", buf, current_text);

                g_idle_add(update_text_buffer_idle, (gpointer)g_strdup(notif));
            }
            else {
                g_idle_add(update_text_buffer_idle, (gpointer)g_strdup(buf));
            }
        }

        if(strcmp(buf, "Eroare la logare.") == 0) g_idle_add((GSourceFunc)showErrorAutLabel, "Eroare
la logare.");
        else if(strcmp(buf, "Deconectat.") == 0) g_idle_add((GSourceFunc)show_menu_window, NULL);
        else if(strcmp(buf, "Quitting...") == 0) break;
        else if(strcmp(buf, "Logat.") == 0 || strcmp(buf, "Inregistrat.") == 0) {
            g_idle_add((GSourceFunc)show_app_window, NULL);
        }
    }
    if (user != NULL) {
        free(user);
        user = NULL;
    }
    if (pass != NULL) {
        free(pass);
        pass = NULL;
    }
    return NULL;
}
```

Figure 12: Citirea mesajelor de la server in client si afisarea lor.

Comenzile care pot fi folosite de client sunt urmatoarele:

- `getMersTren`: Această comandă furnizează informații despre mersul trenurilor la statia curenta pentru ziua respectivă.
- `getSosiri`: Oferă informații despre sosirile planificate pentru urmatoarea ora la stația specificată, inclusiv întârzierile sau estimările de sosire.
- `getPlecari`: Furnizează informații despre plecările planificate pentru urmatoarea ora de la stația specificată, inclusiv eventualele întârzieri.
- `setDelay <nr tren> <nr. poz pt intarziere, negativ pentru mai devreme>`: Permite utilizatorilor să raporteze o întârziere sau o sosire mai devreme pentru un tren specific.
- `cautaTren <nr tren>`: Ajută la găsirea detaliilor despre un tren specific pe baza numărului său.
- `cautaRuta <statie1><statie2>`: Ajută la găsirea unei rute între două stații specifice.
- `login <username><password>`: Permite utilizatorului să se autentifice în sistem introducând numele de utilizator și parola.
- `sign-in <username><password>`: Permite utilizatorului să se inregistreze în sistem introducând numele de utilizator și parola.
- `logout`: Realizează deconectarea utilizatorului curent.
- `quit`: Încheie conexiunea cu serverul și închide aplicația client.
- `statii`: Oferă o listă cu toate statiile disponibile.

5 Scenarii reale de utilizare

Proiectul propus aduce beneficii semnificative în domeniul transportului feroviar, atât pentru călători, cât și pentru operatori. Călătorii pot obține informații precise despre mersul trenurilor, sosiri, plecări și întârzieri, contribuind astfel la o călătorie mai eficientă. De asemenea, posibilitatea de a căuta rute optime între stații sau de a raporta întârzieri oferă utilizatorilor un control mai mare asupra experienței lor. Pentru operatori, sistemul facilitează monitorizarea în timp real a traficului feroviar și gestionarea eficientă a stațiilor.

6 Concluzii

În concluzie, proiectul "Mersul Trenurilor" reprezintă o soluție eficientă și utilă pentru îmbunătățirea experienței călătorilor și optimizarea operațiunilor în domeniul transportului feroviar. Eventuale îmbunătățiri pot include introducerea

unui rol de administrator cu beneficii distincte față de un utilizator simplu (modificarea datelor, gestionarea conturilor utilizatorilor etc., permisiuni pentru notificarea întârzierilor), și adăugarea a mai multor butoane/comenzi precum afișarea trenului cu timp optim de la stația x la stația y, detalii despre prețuri, locuri disponibile în tren etc., care să faciliteze accesul la informații suplimentare și să extindă funcționalitățile oferite de aplicație.

7 Bibliografie

1. https://ro.wikipedia.org/wiki/Transmission_Control_Protocol
2. https://www.geeksforgeeks.org/multithreading-in-c
3. https://www.invata-programare.ro/article/tcp-si-udp-ce-sunt-si-care-sunt-diferentele
4. https://zetcode.com/db/sqlitec
5. https://profs.info.uaic.ro/computernetworks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c
6. https://profs.info.uaic.ro/computernetworks/files/NetEx/S12/ServerConcThread/cliTcpNr.c
7. https://www.w3schools.com/xml/xml_syntax.asp
8. https://docs.gtk.org/gtk3/