Poshet (A) - Raport Tehnic

Matei Ciobanu - 2B1

December 10, 2024

1 Introducere

Proiectul Poshet presupune dezvoltarea unui client de e-mail cu interfață grafică, care să comunice cu două servere: unul care implementează protocolul POP3 pentru recepționarea și gestionarea mesajelor (download, view, delete) și altul care implementează protocolul SMTP pentru transmiterea acestora (send, forward, reply, filter). Clientul pe care îl voi implementa va fi capabil să gestioneze inclusiv atașamentele mesajelor, conform standardului MIME, pentru a extrage și a vizualiza fișierele atașate.

Scopul acestui proiect este de a furniza o experiență interactivă care permite utilizatorului să trimită și să primească mesaje e-mail, să gestioneze atașamentele acestora și să aplice funcții de organizare a mesajelor, într-o aplicație cu o interfață grafică atractivă și intuitivă. **Obiectivele** principale ale proiectului includ realizarea comunicării între client și servere folosind TCP concurent cu prethreading, respectarea standardelor POP3, SMTP și MIME în comunicare și manipularea mailurilor utilizând o bază de date specifică (SQLite).

2 Tehnologii Aplicate

În realizarea acestui proiect, am ales să utilizez limbajul **C++** pentru implementare, în combinație cu tehnologii specifice pentru a dezvolta o aplicație bazată pe comunicarea prin **TCP** și **gestionarea concurentă a clienților**. Alegerea **TCP** în detrimentul **UDP** se datorează nevoii de fiabilitate și securitate în contextul aplicației de e-mail. Protocolul **TCP** garantează livrarea datelor în ordinea corectă și permite detectarea și corectarea erorilor de transmisie, fiind esențial pentru integritatea și ordinea mesajelor.

Pentru a răspunde cerințelor de performanță și scalabilitate, am implementat un model **concurent cu prethreading**. Această abordare presupune alocarea unui fir de execuție pentru fiecare client care se conectează la server, permițând gestionarea multiplelor conexiuni într-un mod concurent. **Prethreading-ul** contribuie semnificativ la îmbunătățirea performanței, reducând riscurile de blocaje și asigurând procesarea rapidă a cererilor, chiar și atunci când serverul are de gestionat un număr mare de clienți simultan.

În ceea ce privește comunicarea între client și server, am ales să folosesc **BSD Sockets**, o metodă standardizată și eficientă pentru a crea conexiuni între procese. Aceasta îmi va permite să realizez un schimb de mesaje simplu și clar între server și client, respectând protocoalele **POP3** și **SMTP**.

Pentru stocarea informațiilor despre utilizatori, mesajele primite și trimise, precum și alte date necesare funcționării corecte a aplicației, am ales utilizarea bazei de date **SQLite**. Această alegere s-a bazat pe avantajele de integrare ușoară și performanță, întrucât **SQLite** nu necesită un server separat. Aceasta va facilita gestionarea datelor într-o manieră rapidă și simplă, adaptată cerințelor aplicației.

3 Structura Aplicației

Aplicația POSHET este divizată în două componente principale: serverele POP3 și SMTP și clientul, care sunt conectate prin intermediul socket-urilor. Programarea cu socket-uri reprezintă o modalitate de a conecta două noduri într-o rețea pentru a comunica între ele. Un socket (nod) ascultă pe un port specific la o adresă IP, în timp ce celălalt socket inițiază conexiunea pentru a se conecta la celălalt. Serverul formează socket-ul ascultător, iar clientul se conectează la server.

3.1 Server POP3

Acesta va gestiona cererile de descărcare a e-mailurilor de la client. După autentificarea utilizatorului, clientul va trimite comenzi pentru a primi informații despre mesaje, pentru a citi mesajele și pentru a le șterge. Serverul POP3 va răspunde la aceste comenzi conform standardului POP3, asigurând o comunicare eficientă între client și server.

Comenzile principale:

1. USER: Comanda folosită pentru identificarea utilizatorului pe serverul POP3.
2. PASS: Comanda folosită pentru autentificarea utilizatorului pe serverul POP3.
3. STAT: Oferă informații despre numărul de mesaje și dimensiunea totală a maildrop-ului.
4. LIST: Oferă informații detaliate despre mesajele din maildrop.
5. RETR: Permite recuperarea unui mesaj specific din maildrop.
6. DELE: Marchează un mesaj pentru ștergere.
7. QUIT: Închide sesiunea POP3, ștergând mesajele marcate pentru ștergere.

Flow client-server POP3:

Protocolul POP3 se bazează pe stări de sesiune pentru a permite un control eficient asupra e-mailurilor. Sesiunea trece prin trei stadii:

1. Stadiul de Autorizare (Authorization State): La început, serverul trimite un mesaj de salut și așteaptă autentificarea clientului prin comenzile USER și PASS. După autentificare, clientul poate începe să interacționeze cu serverul.

2. Stadiul de Tranzacție (Transaction State): După autentificare, clientul poate începe să interacționeze cu mesajele din maildrop prin comenzi precum STAT, LIST, RETR și DELE. Serverul răspunde cu informațiile necesare și permite gestionarea mesajelor.

3. Stadiul de Actualizare (Update State): La finalizarea sesiunii, clientul trimite comanda QUIT, iar serverul șterge mesajele marcate pentru ștergere și închide conexiunea.

3.2 Server SMTP

Acesta va gestiona cererile de trimitere a e-mailurilor de la client. După autentificarea utilizatorului, clientul va trimite comenzi pentru a trimite mesaje, a verifica starea cozii de mesaje și a închide sesiunea. Serverul SMTP va răspunde la aceste comenzi conform standardului SMTP, asigurând o comunicare eficientă între client și server.

Comenzile principale:

1. **HELO**: Comanda folosită pentru a iniția sesiunea de comunicare între client și server.

2. **MAIL FROM**: Specifică adresa de expediere a mesajului.

3. **RCPT TO**: Specifică adresa de destinație pentru mesajul e-mail.

4. **DATA**: Permite clientului să trimită conținutul mesajului, inclusiv anteturile și corpul.

5. **RSET**: Resetează sesiunea curentă și șterge orice informație parțială despre mesaje trimise.

6. **QUIT**: Închide sesiunea SMTP.

Ambele servere vor răspunde la comenzile clientului, asigurând o comunicare eficientă și conformă cu protocoalele POP3 și SMTP.

3.3 Client

Clientul POSHET va permite utilizatorului să interacționeze cu serverele POP3 și SMTP printr-o interfață grafică intuitivă. Clientul va trimite comenzile corespunzătoare pentru descărcarea mesajelor, trimiterea e-mailurilor, selectarea și ștergerea mesajelor și organizarea e-mailurilor. Acesta va primi răspunsuri de la servere și va actualiza interfața cu informațiile primite, precum subiectul mesajului, conținutul și eventualele atașamente.

3.4 Comunicarea între Client și Server

Comunicarea între client și server se va face prin utilizarea socket-urilor și a protocoalelor TCP. Protocolul TCP a fost ales în locul UDP datorită fiabilității și securității superioare, esențiale într-o aplicație de tip client-server, precum POSHET. TCP garantează livrarea datelor în ordinea corectă și permite detectarea și corectarea erorilor de transmisie.

Serverele POP3 și SMTP vor utiliza BSD Sockets pentru a facilita conexiunea de rețea.

3.5 Gestionarea Concurentă cu Prethreading

Pentru a asigura gestionarea eficientă a conexiunilor multiple, aplicația va implementa un model concurent cu *prethreading*. Fiecare client care se conectează la server va fi gestionat de un fir de execuție separat. Această abordare ajută la creșterea performanței prin alocarea unui fir de execuție pentru fiecare client, permițând serverelor să răspundă rapid la mai multe cereri simultan fără a afecta performanța generală a aplicației.

3.6 Baza de Date SQLite

Pentru stocarea informațiilor despre utilizatori, mesajele primite și trimise, și alte date necesare funcționării aplicației, vom utiliza o bază de date SQLite. SQLite este o soluție ușor de integrat și performantă, care nu necesită un server separat, fiind ideală pentru aplicațiile client-server de tipul POSHET. Baza de date va permite gestionarea ușoară a mesajelor și conturilor de utilizator, contribuind la buna funcționare a aplicației.

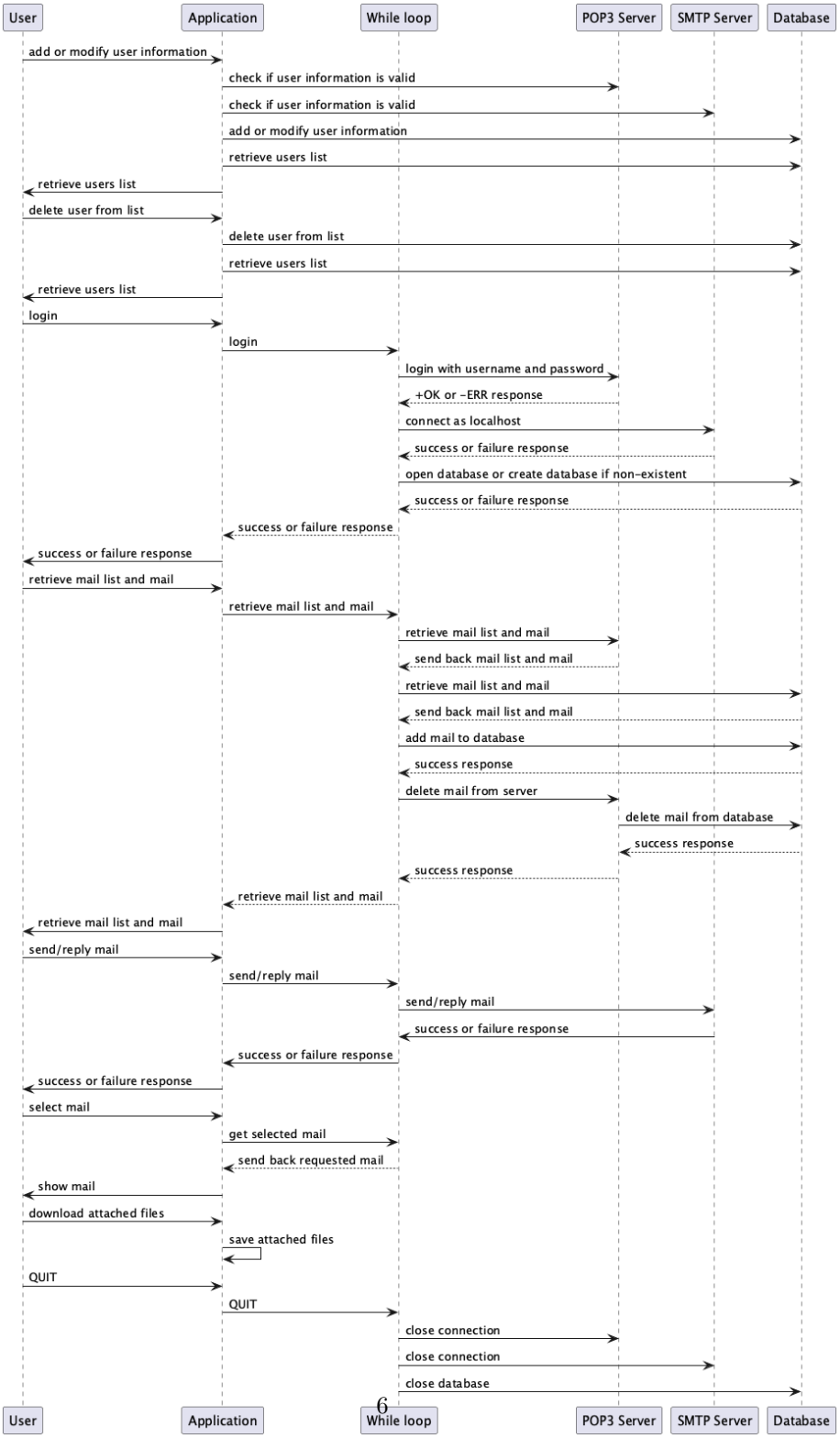
3.7 Fluxul de Date și Gestionarea Sesiunilor

Serverele POP3 și SMTP vor respecta stadiile sesiunii, conform protocoalelor RFC. După stabilirea conexiunii, serverul POP3 va trimite un mesaj de salut, iar clientul va transmite comenzile necesare pentru autentificare și descărcarea mesajelor. Serverul va răspunde fie cu mesaje de succes, fie cu mesaje de eroare, în funcție de comanda trimisă.

În ceea ce privește serverul SMTP, acesta va răspunde la comenzile de trimitere a mesajelor, confirmând livrarea acestora sau indicând erorile care au apărut. Clientul va interacționa cu aceste servere printr-o interfață grafică

simplă, care va permite trimiterea și primirea mesajelor, gestionarea atașamentelor și organizarea e-mailurilor.

Diagrama de mai jos include componentele de bază, fluxurile de date dintre ele și modul în care sunt gestionate în timpul funcționării aplicației.


User
Application
While loop
POP3 Server
SMTP Server
Database
add or modify user information
check if user information is valid
check if user information is valid
add or modify user information
retrieve users list
retrieve users list
delete user from list
delete user from list
retrieve users list
retrieve users list
login
login
login with username and password
+OK or -ERR response
connect as localhost
success or failure response
open database or create database if non-existent
success or failure response
success or failure response
success or failure response
retrieve mail list and mail
retrieve mail list and mail
retrieve mail list and mail
send back mail list and mail
retrieve mail list and mail
send back mail list and mail
add mail to database
success response
delete mail from server
delete mail from database
success response
success response
retrieve mail list and mail
retrieve mail list and mail
send/reply mail
send/reply mail
send/reply mail
success or failure response
success or failure response
success or failure response
select mail
get selected mail
send back requested mail
show mail
download attached files
save attached files
QUIT
QUIT
close connection
close connection
close database
User
Application
While loop
POP3 Server
SMTP Server
Database

4 Aspecte de Implementare

```
sqlite3* db;
sqlite3_stmt* stmt;
const char* dbPath = "ProiectRetele.db";

if (sqlite3_open(dbPath, &db) != SQLITE_OK)
{
    std::cerr << "Failed to open database: " << sqlite3_errmsg(db) << std::endl;
    return false;
}
```

În cadrul acestei porțiuni de cod, deschidem baza de date prin intermediul funcției sqlite3 open pentru ca serverul să aibă acces la datele stocate în baza de date. Această operație permite autentificarea utilizatorilor și gestionarea mesajelor prin intermediul unei baze de date locale SQLite.

```
int serverSocket = socket(AF_INET, SOCK_STREAM, 0);
if (serverSocket < 0)
{
    std::cerr << "Failed to create socket.\n";
    return;
}
```

În această secțiune, se creează socket-ul, specificând tipul de protocol și familia sa de adrese. În cazul proiectului ales de mine, se poate observa că am creat un socket TCP aparținând familiei de adrese IPv4, prin flag-urile AF INET și SOCK STREAM. Aceasta permite stabilirea unei conexiuni sigure și fiabile între client și server.

```cpp
// Creăm un pool de thread-uri
std::vector<std::thread> threadPool;
for (int i = 0; i < THREAD_POOL_SIZE; ++i)
{
    threadPool.emplace_back(workerThread);
}

while (true)
{
    sockaddr_in clientAddr;
    socklen_t clientLen = sizeof(clientAddr);
    int clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddr, &clientLen);
    if (clientSocket < 0)
    {
        std::cerr << "Failed to accept connection.\n";
        continue;
    }

    // Adăugăm conexiunea în coadă
    {
        std::unique_lock<std::mutex> lock(queueMutex);
        clientQueue.push(clientSocket);
    }
    queueCondition.notify_one();
}
```

În cadrul acestei porțiuni de cod, deservim în mod concurent clienții prin intermediul thread-urilor. Serverul rulează într-o buclă infinită, ceea ce înseamnă că va aștepta și deservi clienți în mod continuu. La început, este creat un *pool* de thread-uri, stocat într-un vector, pentru a păstra toate thread-urile active ale serverului. Apoi, serverul rulează o buclă infinită pentru a accepta conexiuni de la clienți. Odată ce un client se conectează, funcția 'accept' returnează un descriptor de socket unic prin care ne referim la această conexiune. Ulterior, adăugăm conexiunea într-o coadă sincronizată printr-un mutex, astfel încât doar un singur proces poate adăuga la coadă la un moment dat. Atunci când o conexiune este adăugată la coadă, unul dintre thread-urile din *pool* este notificat, iar acesta preia conexiunea din coadă și o procesează folosind o funcție specifică. Astfel, fiecare thread din *pool* poate lucra cu câte un client independent, asigurând concurența serverului TCP.

```
// Funcția executată de fiecare thread din pool
void workerThread()
{
    while (true)
    {
        int clientSocket;

        // Așteptăm conexiuni noi în coadă
        {
            std::unique_lock<std::mutex> lock(queueMutex);
            queueCondition.wait(lock, [] { return !clientQueue.empty(); });
            clientSocket = clientQueue.front();
            clientQueue.pop();
        }

        // Gestionăm clientul
        handlePop3Client(clientSocket);
    }
}
```

Funcția 'workerThread' gestionează conexiunile clienților într-un mediu concurent, fiind utilizată de fiecare thread în parte. Aceasta rulează într-o buclă infinită, așteptând conexiuni noi adăugate într-o coadă sincronizată. Cu ajutorul unui mutex și al unei condiții variabile, funcția asigură accesul exclusiv la coadă, preluând socket-ul conexiunii unui client și eliminându-l din coadă. După preluare, conexiunea este procesată utilizând funcția 'handlePop3Client', iar thread-ul revine la starea de așteptare pentru a gestiona următorul client. Acest mecanism eficient permite procesarea simultană a mai multor conexiuni fără a recrea constant thread-uri.

```cpp
while (true)
{
    std::cout << "\nSelect server to interact with:\n";
    std::cout << "1. POP3 Server\n";
    std::cout << "2. SMTP Server\n";
    std::cout << "3. Quit\n";
    std::cout << "Choice: ";

    int choice;
    std::cin >> choice;
    std::cin.ignore(); // Clear newline from input buffer

    if (choice == 1)
    {
        if (!pop3Thread.joinable())
        {
            pop3Thread = std::thread(handleServerConnection, "POP3", "127.0.0.1", 8110);
            pop3Thread.join(); // Wait for the thread to finish
        }
    }
    else if (choice == 2)
    {
        if (!smtpThread.joinable())
        {
            smtpThread = std::thread(handleServerConnection, "SMTP", "127.0.0.1", 8025);
            smtpThread.join(); // Wait for the thread to finish
        }
    }
    else if (choice == 3)
    {
        std::cout << "Exiting client...\n";
        break;
    }
    else
    {
        std::cout << "Invalid choice. Try again.\n";
    }
```

Această bucată de cod implementează un meniu interactiv pentru client, permițând utilizatorului să selecteze serverul cu care dorește să interacționeze (POP3 sau SMTP) sau să părăsească aplicația. În funcție de alegerea utilizatorului, un thread este creat pentru a se conecta la serverul respectiv folosind funcția 'handleServerConnection'. Dacă utilizatorul selectează **Quit**, aplicația afișează un mesaj și oprește bucla, terminând execuția. Codul verifică dacă thread-urile POP3 și SMTP sunt deja active înainte de a crea un nou thread pentru fiecare alegere. Acest mecanism asigură o execuție controlată și sincronizată a operațiunilor clientului.

4.1 Descrierea a 2 scenarii reale de utilizare:

1. Verificarea mesajelor primite prin POP3

Un utilizator pornește aplicația client pentru a-și verifica emailurile. Din meniul interactiv, selectează conectarea la serverul POP3. După conectare, se autentifică folosind numele de utilizator și parola, serverul confirmând identitatea acestuia. Utilizatorul folosește comenzile disponibile pentru a vedea câte mesaje are în inbox, pentru a lista detalii despre acestea sau pentru a citi conținutul

unui mesaj specific. După ce a verificat toate mesajele necesare, utilizatorul încheie conexiunea trimițând comanda **QUIT**.

2. Trimiterea unui email prin SMTP

Un utilizator dorește să trimită un email unui coleg. Acesta pornește aplicația client și selectează conectarea la serverul SMTP din meniul principal. După stabilirea conexiunii, introduce comanda **HELO** pentru a iniția comunicarea. Specifică adresa de email a expeditorului prin comanda **MAIL FROM**, urmată de adresa destinatarului folosind comanda **RCPT TO**. La comanda **DATA**, redactează subiectul și corpul emailului, iar apoi finalizează mesajul introducând comanda de încheiere. Serverul confirmă trimiterea reușită a mesajului, iar utilizatorul închide sesiunea cu comanda **QUIT**.

5 Concluzii

Proiectul POSHET oferă o experiență interactivă și utilă pentru utilizatorii care doresc să gestioneze eficient e-mailurile, atât pentru trimitere, cât și pentru recepționare. Totuși, există câteva îmbunătățiri posibile care ar putea aduce un plus de valoare aplicației: integrarea unei funcționalități avansate de filtrare și organizare a mesajelor, implementarea unei autentificări mai complexe pentru creșterea securității utilizatorilor, precum și realizarea unei interfețe grafice mai moderne și atractive pentru clientul POSHET. Aceste optimizări ar contribui la creșterea funcționalității și a satisfacției utilizatorilor în utilizarea aplicației.

6 Bibliografie

1. https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php
2. https://ro.wikipedia.org/wiki/POP3
3. https://ro.wikipedia.org/wiki/SMTP
4. https://en.wikipedia.org/wiki/MIME
5. https://en.wikipedia.org/wiki/Transmission_Control_Protocol
6. https://edu.info.uaic.ro/computer-networks/files/7rc_ProgramareaInReteaIII_En.pdf
7. https://www.geeksforgeeks.org/thread-in-operating-system/
8. https://www.geeksforgeeks.org/socket-programming-cc/
9. https://www.geeksforgeeks.org/what-is-transmission-control-protocol-tcp/
10. https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/
11. https://sqlite.org/docs.html