Rețele de Calculatoare – VirtualSoc

Moise Ioana-Simina 2B2

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

1 Introducere

În acest Raport Tehnic vreau să prezint viziunea generală a proiectului ales de mine , acesta fiind "VirtualSoc". Obiectivul este de a simula o rețea socială care permite utilizatorilor să interacționeze și să partajeze conținut, respectând drepturile de acces.Proiectul include funcționalități precum autentificarea, adăugarea și ștergerea de postări, știri sau prieteni, precum și vizualizarea istoricului evenimentelor. Acest proiect combină programarea avansată în limbajul C, manipularea fișierelor XML și programarea socket pentru a implementa un sistem complet de comunicație între server și client.

2 Tehnologii Aplicate

Tehnologiile utilizate în acest proiect includ programarea socket pentru comunicații de rețea, unde sunt folosite biblioteci POSIX pentru crearea și gestionarea conexiunilor TCP. Codul implementează funcții specifice precum socket, bind, listen, accept, connect, send și recv pentru stabilirea și gestionarea comunicațiilor între server și client. Un alt aspect important este utilizarea multithreading-ului, prin biblioteca POSIX Threads (pthread), care permite serverului să gestioneze simultan mai mulți clienți. Mutex-urile sunt utilizate pentru a sincroniza accesul la resursele partajate, cum ar fi fișierele XML. Pentru gestionarea datelor, proiectul utilizează biblioteca libxml2 pentru manipularea fișierelor XML. Aceasta permite parsarea, crearea, modificarea și salvarea documentelor XML. Funcții precum xmlParseFile, xmlDocGetRootElement, xmlNewNode, xmlNewChild și xmlSetProp sunt folosite pentru gestionarea nodurilor și atributelor XML. Salvarea documentelor XML se realizează cu formatare indentată pentru o lizibilitate mai bună.Pentru gestionarea clientilor am folosit o lista simpla inlantuita pentru stocarea clientilor pe care am gestionat-o cu ajutorul unui struct Client care are socket: Descriptorul de socket asociat conexiunii clientului,nume: Numele utilizatorului,tip: Tipul utilizatorului (ex. "admin", "user", etc.),profil: Profilul utilizatorului ("public" sau "privat"),authenticated: Flag care indică dacă utilizatorul este autentificat,next: Pointer către următorul client în listă (implementare de listă simplă înlănțuită). Gestionarea stringurilor este esențială în procesarea comenzilor și a datelor.Funcția personalizată strtrim este folosită pentru eliminarea spațiilor suplimentare.Proiectul folosește structuri ierarhice pentru organizarea

informațiilor în fișiere XML, oferind un mod eficient de stocare și accesare a datelor despre utilizatori, prieteni și evenimente. Protocolul de comunicare între server și client este text-based, ceea ce înseamnă că comenzile și răspunsurile sunt trimise și recepționate sub formă de text.

3 Structura Aplicației

Aplicația implementată este un sistem client-server care simulează o rețea socială simplificată. Serverul gestionează conexiunile și cererile clienților, iar interacțiunile sunt procesate printr-un protocol definit specific pentru această aplicație. Serverul acceptă conexiuni simultane de la clienți, stochează informațiile utilizatorilor în memorie și procesează cererile într-un mod multi-threaded, fiecare client având un thread dedicat. Serverul implementează următoarele funcționalități: autentificare, gestionare prieteni, postare mesaje, trimitere știri, gestionare profil utilizator trimitere mesaje private și deconectare. Clientul permite utilizatorului să introducă comenzi și să interacționeze cu serverul prin intermediul unui terminal. Acesta afișează răspunsurile primite de la server și garantează funcționarea de bază printr-un set de comenzi introduse manual.

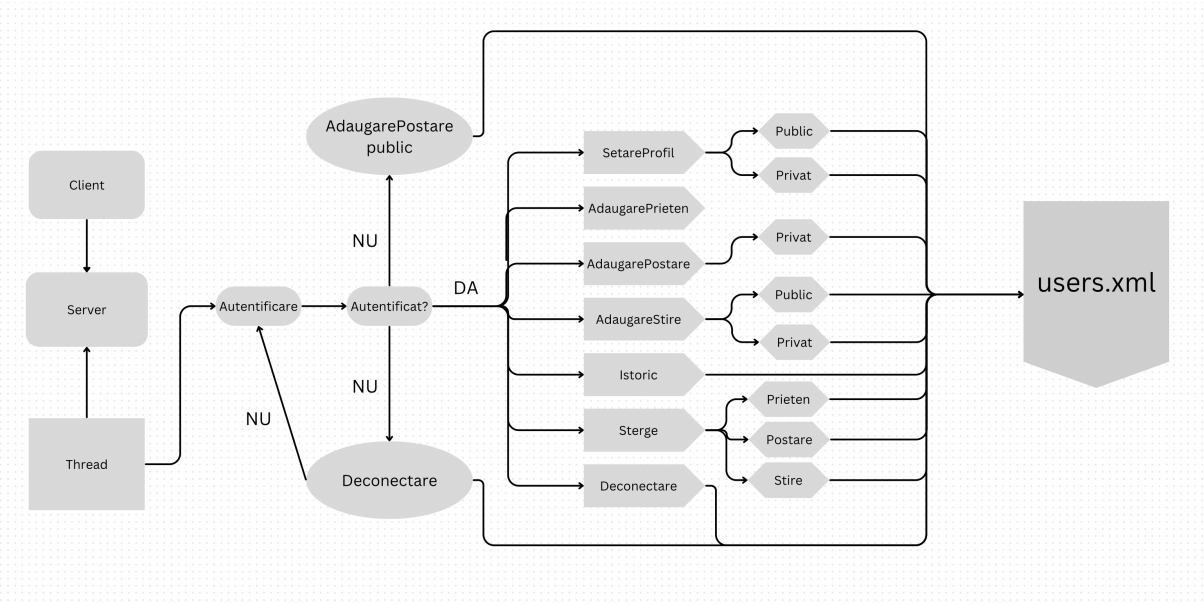

Figure 1: Diagramă UML

4 Aspecte de Implementare

În codul serverului prezentat, utilizarea threadurilor și mutexurilor joacă un rol esențial în gestionarea corectă și eficientă a conexiunilor multiple și a concurenței dintre clienți. În implementare, funcția `pthread_create` este folosită pentru a crea un thread dedicat fiecărui client:

```
    pthread_t thread_id;
    int *arg = malloc(sizeof(int));
    *arg = client_sock;
    if (pthread_create(&thread_id, NULL, handle_client, arg) != 0) {
        perror("Eroare la crearea thread-ului");
        free(arg);
        close(client_sock);
    }
    pthread_detach(thread_id);
}
```

Mutexurile (`pthread_mutex_t`) sunt utilizate pentru a sincroniza accesul la resurse partajate, cum ar fi fișierul XML care stochează informațiile utilizatorilor. Acestea asigură că doar un singur thread poate accesa sau modifica resursa respectivă la un moment dat, prevenind condițiile de cursă și coruperea datelor. Un exemplu de utilizare a mutexurilor în cod este protejarea operațiunilor de citire/scriere în fișierul XML:

```
pthread_mutex_lock(&db_mutex);
add_user_event("users.xml", nume, tip, profil,
               "Autentificare utilizator", "login");
pthread_mutex_unlock(&db_mutex);
```

Mutexul (`db_mutex`) este inițializat la începutul programului și distrus la încheierea acestuia. Utilizarea threadurilor și mutexurilor în acest server permite procesarea eficientă și sigură a cererilor multiple ale clienților. Threadurile gestionează concurența prin crearea unui context separat pentru fiecare client, iar mutexurile asigură integritatea datelor și elimină riscurile asociate accesării simultane a resurselor partajate.

5 Concluzii

Concluziile proiectului evidențiază succesul în implementarea unui sistem de comunicație client-server care utilizează tehnologii avansate pentru gestionarea conexiunilor, manipularea datelor și sincronizarea accesului. Serverul este capabil să gestioneze mai mulți clienți simultan, utilizând multithreading pentru a asigura performanța și scalabilitatea. Mecanismele de sincronizare prin mutex-uri previn conflictele între procese și asigură integritatea datelor partajate, mai ales în contextul accesului concurent la fișierele XML. Fișierele XML s-au dovedit a fi o soluție eficientă pentru stocarea informațiilor structurale despre utilizatori, prieteni și evenimente. Utilizarea bibliotecii libxml2 a permis manipularea flexibilă și precisă a acestor date, asigurând în același timp o organizare clară și lizibilă. Protocolul de comunicare bazat pe text între server și client este intuitiv și extensibil, oferind utilizatorilor un set clar de comenzi pentru interacțiune. Această abordare simplifică procesul de testare și depanare, deoarece fluxul de date este ușor de urmărit. Funcționalitățile oferite, cum ar fi autentificarea, adăugarea și gestionarea prietenilor, postărilor și știrilor, demonstrează flexibilitatea și complexitatea sistemului. În plus, înregistrarea evenimentelor și istoricul activităților adaugă un nivel suplimentar de transparență și utilitate.

6 Referințe Bibliografice

1. Universitatea "Alexandru Ioan-Cuza" din Iaşi - Cursuri și laboratoare - Rețele de calculatoare: https://edu.info.uaic.ro/computer-networks/cursullaboratorul.php

2. Overleaf - Cum să incluzi cod în LaTeX: https://www.overleaf.com/learn/latex/Code_listing

3. Exemplu de cod pentru un server pre-thread din cursul UAIC: https://edu.info.uaic.ro/computernetworks/files/NetEx/S12/ServerPreThread/servTcPreTh.c

4. GeeksforGeeks - Funcții pentru thread-uri în C/C++: https://www.geeksforgeeks.org/thread-functions-in-c-c/

5. GeeksforGeeks - Utilizarea funcției 'snprintf' în biblioteca C: https://www.geeksforgeeks.org/snprintf-c-library/ 6. Exemplu de server TCP concurent care deserveste clientii prin crearea unui thread pentru fiecare client: https://edu.info.uaic.ro/computer-networks/files/NetEx/S12/ServerConcThread/servTcpConcTh2.c

7. Exemplu de client TCP: https://edu.info.uaic.ro/computer-networks/files/NetEx/S12/ServerConcThread/cliTcpNr.c