Raport tehnic

WebS

January 2024

1 Introducere

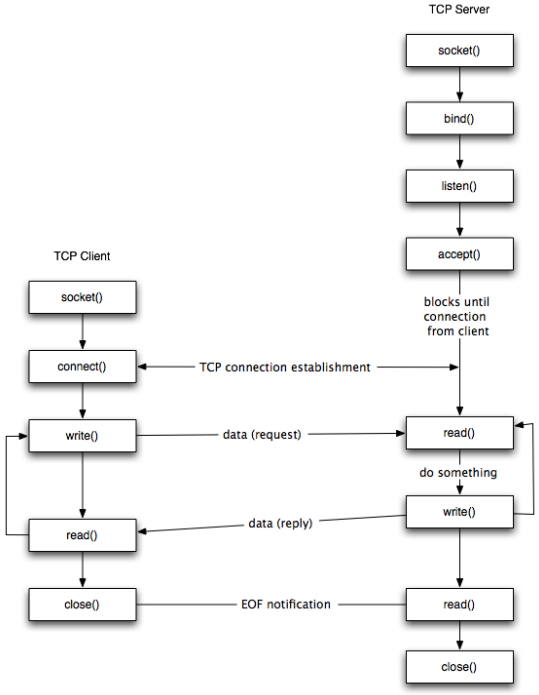
WebS implementează un client si un server web de baza care vor comunica folosind protocolul TCP. Serverul contine cateva pagini web (fisiere .html simple) si pune la dispozitia clientilor urmatoarele comenzi: login (autentificarea la server), list (furnizeaza lista cu fisierele continute), get (copie o pagina web de pe server), add (uploadeaza pe server o pagina web), delete (sterge o pagina web de pe server). Clientul interactioneaza cu serverul folosind comenzile oferite.

2 Tehnologiile utilizate

Pentru a implementa HTTP, ne pasă doar de al 4-lea Layer: Transport Layer. În Transport Layer folosim în principal protocolul de control al transmisiei (TCP) pentru a implementa un server HTTP. Pentru comunicarea dintre client și server vom folosi TCP. TCP (Protocolul de Control al Transmisiei) este cel mai larg folosit în lume și se referă la o familie de protocoale înrudite, toate proiectate pentru a transfera informații.Toate părțile protocolului TCP au anumite sarcini, cum ar fi trimiterea de scrisori electronice, transfer de fișiere (ceea ce vom folosi cu precădere în aplicație),livrarea de servicii de logare la distanță, dirijarea de mesaje, sau manipularea căderilor de rețea.

3 Arhitectura aplicației

Conexiunea dintre server si clienți se va face prin socket-uri. Modelul de comunicare dintre acestea este TCP server/client concurent regăsit în imaginea de mai jos:



Mai mulți clienți se vor putea conecta la server, simultan. Serverul este unul concurent, ceea ce inseamna ca are abilitatea de a lucra cu clienți multiplii folosind "thread-uri". Fiecare client are un thread separat.

4 Detalii de implementare

Cum va incepe serverul:

1. Crearea și configurarea serverului:
   - Se creează un socket folosind socket()
   - Se pregătește structura pentru server (server) cu informații precum familia de adrese, adresa IP și portul.
2. Legarea serverului la adresa și portul specificate:
   - Se utilizează bind() pentru a asocia socketul cu adresa și portul specificate.
   - Dacă există o eroare, se afișează un mesaj și programul se încheie.
3. Ascultarea pentru conexiuni de la clienți:
   - Se folosește listen() pentru a permite serverului să asculte conexiuni de la clienți.

- Dacă există o eroare, se afișează un mesaj și programul se încheie.

4. Acceptarea conexiunilor de la clienți:
   - Se intră într-o buclă infinită care așteaptă și acceptă conexiuni de la clienți folosind accept().
   - După acceptarea cu succes a unei conexiuni, se creează un nou thread pentru a trata interacțiunea cu clientul.
5. Inițierea unui thread pentru fiecare client:
   - Un nou obiect thData este alocat pentru fiecare client și inițializat cu informații precum idThread, descriptorul de client (cl), și starea de autentificare (authenticated).
   - Un nou thread este creat, iar funcția treat este apelată pentru a începe gestionarea clientului în mod concurent.
6. Funcția treat și gestionarea clientului:
   - Se afișează un mesaj cu privire la așteptarea mesajului de la client.
   - Se dezactivează thread-ul curent pentru a-l face detasabil.
   - Funcția raspunde este apelată pentru a gestiona interacțiunea cu clientul.
7. Funcția raspunde și gestionarea comenzilor:
   - Într-o buclă infinită, se așteaptă comenzi de la client folosind read().
   - Dacă se întâmpină o eroare la citire, se afișează un mesaj și bucla se încheie.
   - Dacă comanda este "exit", bucla se încheie, dar conexiunea nu este închisă.
   - Pentru comenzi precum "login", "list", "get", "add", "delete", se apelează funcțiile corespunzătoare (handle-login, handle-list, etc.).
   - Răspunsurile sau mesajele de eroare rezultate sunt trimise înapoi la client folosind write().
8. Închiderea conexiunii cu clientul:

   După încheierea tratării unui client, thread-ul curent se încheie, iar resursele asociate conexiunii sunt eliberate.

```
if (strncmp(command, "login", 5) == 0) {
    char username[256];
    char parola[256];
    sscanf(command, "login %s %s", username, parola);
    handle_login(tdL.cl, username, parola, &tdL);
    if (tdL.authenticated == 1) {
        printf("Autentificare cu succes!\n");
    } else {
        printf("Autentificare esuata!\n");
        char message[] = "Autentificare esuata!\n";
        if (write(tdL.cl, message, strlen(message)) <= 0) {
            printf("[Thread]Eroare la write() catre client.\n");
        } else {
            printf("Mesajul a fost trasmis cu succes.\n");
        }
        //break;

} else if (strcmp(command, "list") == 0) {
    handle_list(tdL.cl);
} else if (strncmp(command, "get", 3) == 0) {
    char page_name[256];
    sscanf(command, "get %s", page_name);
    handle_get(tdL.cl, page_name);
} else if (strncmp(command, "add", 3) == 0) {
    char page_name[256];
    char page_content[1024];
    sscanf(command, "add %s %[^\n]s", page_name, page_content);
    handle_add(tdL.cl, page_name, page_content);
} else if (strncmp(command, "delete", 6) == 0) {
    char page_name[256];
    sscanf(command, "delete %s", page_name);
    handle_delete(tdL.cl, page_name);
} else {
    perror("Eroare la tot.\n");
```

5 Concluzii

Proiectul este un server simplu care gestionează interacțiunea cu clienții într-un mod concurent. Acesta oferă o interfață simplificată pentru utilizatori, permițându-le să autentifice, să vizualizeze și să manipuleze pagini web stocate. O posibila imbunatatire la acest proiect ar putea fi in cazul comenzii get, deschizand html-ul intr-un browser.

6 Bibliografie

- https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/ServerConcThread/cliTcpNr.c
- https://profs.info.uaic.ro/ computernetworks/files/NetEx/S12/Server ConcThread/servTcpConcTh2.c