**ACADEMIA TEHNICĂ MILITARĂ “FERDINAND I”**

**FACULTATEA DE SISTEME INFORMATICE ŞI SECURITATE CIBERNETICĂ**

**Specializarea: Calculatoare şi sisteme informatice pentru apărare şi securitate naţională**

A yellow eagle with a crown and a shield

Description automatically generated

Proiectarea sistemelor de operare- Proiect

Mini server Web - cu suport pentru conexiuni multiple, cu thread-uri pooling

Îndrumător:

Slt. Ing. Vaman Adina

Realizat de:

Sd. Sg. Andrei Roxana-Geanina

Sd. Sg. Gruia Ștefan-Bogdan

Grupa C-113E

Cuprins

[1. Introducere 4](#_Toc19619)

[2. Scopul proiectului 4](#_Toc10225)

[3. Descrierea HTTP și a metodelor de cerere 4](#_Toc31929)

[4. Conexiunea cu serverul 6](#_Toc30777)

[5. Descrierea serverului web implementat 8](#_Toc20067)

[Bibliografie 9](#_Toc3930)

**Listă de figuri**

[Image 1 . Basic web server architecture for a static site. Sursa [1] 8](#_Toc10875)

[Image 2 . Basic architecture for a dynamic website. Sursa [1] 8](#_Toc27390)

**Listă de tabele**

[Table 1 . Metode de cerere 5](#_Toc31897)

**Listă de abrevieri**

|  |  |
| --- | --- |
| HTTP | Hypertext Transfer Protocol |
| IP | Internet Protocol |
| TCP | Transmission Control Protocol |
| URL | Uniform Resource Locator |
| MIME | Multipurpose Internet Mail Extensions |
| HTML | Hypertext Markup Language |
| CSS | Cascading Style Sheets |

# Introducere

**Serverele web** permit accesul la informații stocate la distanță și oferă utilizatorilor experiențe interactive prin intermediul aplicațiilor web. Rolul lor este de a primi și procesa cereri de la clienți (de obicei browsere) și de a răspunde în conformitate cu cerințele protocolului HTTP.

Acest proiect își propune să creeze un mini server web folosind limbajul C, proiectat să funcționeze în mediul Ubuntu și integrat într-o aplicație de dezvoltare precum Visual Studio Code. Serverul va fi configurat să asculte pe portul 80 și va răspunde cererilor HTTP specifice, inclusiv GET, POST, DELETE și PUT, pentru a simula funcționarea unui server web real.

Implementarea unui astfel de server oferă oportunitatea de a explora atât concepte de programare la nivel de sistem, cum ar fi manipularea fișierelor și utilizarea rețelelor, cât și noțiuni specifice ale protocolului HTTP. De asemenea, proiectul va include suport pentru gestionarea mai multor tipuri de fișiere și va permite descărcarea acestora, iar în cazul conținutului dinamic va oferi opțiuni pentru apelarea interpretoarelor PHP sau JavaScript.

# Scopul proiectului

Pe parcursul dezvoltării, scopul specific al proiectului include:

1. **Înțelegerea arhitecturii server-client și a protocolului HTTP:** Scopul proiectului este de a explora protocolul HTTP prin implementarea cererilor de bază, conform funcționalităților de tip GET, POST, DELETE și PUT.
2. **Aplicarea practică a conceptelor de networking și sockets în C:** Prin crearea unui server web, acest proiect urmărește să dezvolte abilitățile de programare legate de rețele, în special utilizarea sockets în limbajul C pentru a permite comunicarea dintre server și clienți. Astfel, scopul este să oferim o experiență directă în folosirea funcțiilor socket-urilor pentru deschiderea, ascultarea și gestionarea conexiunilor la portul 80.
3. **Implementarea unor metode de gestionare a conexiunilor simultane:** Un server web real trebuie să fie capabil să gestioneze multiple conexiuni simultane. Prin acest proiect, vom experimenta cu metode precum thread pooling, permițând serverului să răspundă mai multor cereri în paralel.
4. **Dezvoltarea unui server cu suport pentru fișiere și tipuri de conținut multiple:** Un server web modern trebuie să poată servi mai multe tipuri de fișiere (HTML, CSS, JavaScript, imagini, etc.). Proiectul va implementa suport pentru diferite tipuri MIME, permițând descărcarea de fișiere și servirea de conținut static și dinamic. În plus, vor fi implementate apeluri către interpretoare PHP și JavaScript pentru a simula funcționalitatea unui server dinamic.

# Descrierea HTTP și a metodelor de cerere

**Hypertext Transfer Protocol (HTTP)** este un protocol de nivel înalt care definește modul în care sunt transmise informațiile între clienți (de obicei browsere web) și servere pe internet. HTTP este protocolul principal utilizat pentru accesarea paginilor și resurselor web și este bazat pe o arhitectură de tip „request-response” (cerere-răspuns), unde clientul trimite o cerere, iar serverul îi răspunde corespunzător. Protocolul HTTP operează pe portul 80 (în mod implicit).

Cea mai comună utilizare a HTTP constă în descărcarea paginilor HTML, a fișierelor multimedia și a altor resurse, dar și în transmiterea de date de la client la server.

**Metodele de cerere HTTP** definesc acțiunile specifice pe care clientul dorește să le efectueze pe server. Printre cele mai importante și utilizate metode de cerere se numără GET, POST, PUT și DELETE.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metoda | GET | POST | PUT | DELETE |
| Descriere | GET este metoda utilizată cel mai frecvent pentru a solicita resurse de pe un server. Aceasta este o cerere „read-only”, adică scopul său este doar de a accesa și returna informația, fără a efectua modificări. | POST este metoda prin care clientul trimite date către server, pentru a crea sau procesa informații pe server. | PUT este metoda utilizată pentru a crea sau înlocui o resursă pe server. | DELETE este metoda folosită pentru a șterge o resursă specificată pe server. |
| Funcționalitate | Când un client trimite o cerere GET, serverul caută resursa solicitată (de exemplu, o pagină HTML) și returnează conținutul acesteia în corpul răspunsului. | Spre deosebire de GET, POST include un corp de mesaj care conține datele pe care clientul dorește să le transmită serverului (de exemplu, informațiile unui formular). | O cerere PUT include resursa pe care clientul dorește să o creeze sau să o actualizeze. Serverul salvează această resursă la locația specificată de client. | La primirea unei cereri DELETE, serverul identifică resursa și o șterge, dacă există permisiuni suficiente. |
| Aplicații | GET este folosit pentru a accesa pagini web, imagini, fișiere și alte resurse stocate pe server. | POST este utilizat pentru trimiterea datelor sensibile sau pentru operațiuni care necesită schimbări pe server, cum ar fi autentificarea, completarea de formulare sau încărcarea fișierelor. | PUT este des utilizat pentru actualizări de resurse (de exemplu, modificarea unui document sau actualizarea unei înregistrări într-o bază de date). | DELETE este folosit pentru a elimina resurse (cum ar fi fișiere, pagini web sau înregistrări) de pe server. |

Table 1. Metode de cerere

**Structura Cererii și Răspunsului HTTP:**

* **Cererile HTTP** sunt formate dintr-o linie de început (care include metoda, URL-ul și versiunea HTTP), header-uri (cu informații despre client, tipul de conținut și alte detalii), și, uneori, un corp (în special pentru cererile POST sau PUT).
* **Răspunsurile HTTP** includ un cod de stare (ex. 200 OK, 404 Not Found), header-uri care detaliază tipul conținutului și alte informații, și, de asemenea, un corp de mesaj care poate conține resursa solicitată.

# Conexiunea cu serverul

În proiectul nostru de server web, **conexiunea dintre client și server** este fundamentală pentru a permite transmiterea și recepționarea de date prin intermediul cererilor HTTP. Această conexiune este stabilită prin utilizarea **socket-urilor**, care permit un schimb de informații în mod bidirecțional, de la browserul clientului către serverul nostru și invers. În mod concret, serverul nostru va folosi un **socket TCP**, care va rămâne deschis pentru a asculta cereri HTTP pe portul 80. Browserul va trimite cereri de tip GET, POST, PUT și DELETE, iar serverul nostru va răspunde la fiecare dintre acestea prin intermediul socket-ului deschis, gestionând datele corespunzătoare fiecărei metode.

**Configurarea și Utilizarea Socket-urilor:**

1. ***Crearea socket-ului serverului***

* Vom începe prin a crea un socket TCP de tip SOCK\_STREAM, care va permite comunicarea stabilă și fiabilă între server și client.
* Crearea socket-ului se va face utilizând funcția socket(), care va inițializa un nou socket configurat pentru protocolul IPv4 (AF\_INET) și TCP.

1. ***Asocierea socket-ului la adresa IP și portul 80***

* După crearea socket-ului, acesta va fi asociat adresei IP locale și portului 80 utilizând funcția bind(). Această asociere permite serverului să primească cereri HTTP de la clienți la adresa specificată.
* Într-un mediu local, vom folosi adresa IP 127.0.0.1 sau localhost pentru testare, dar serverul poate fi configurat pentru a asculta pe orice interfață de rețea disponibilă.

1. ***Ascultarea pentru cereri noi (listen)***

* După asocierea socket-ului, serverul va intra în modul de ascultare, utilizând funcția listen(), care îi permite să aștepte conexiuni din partea clienților.
* Această funcție va configura serverul nostru pentru a accepta multiple conexiuni simultane, astfel încât să poată răspunde mai multor cereri HTTP de la diverse browsere în același timp.

1. ***Acceptarea conexiunilor și crearea de fire de execuție***

* De fiecare dată când un client inițiază o conexiune, serverul va accepta conexiunea cu ajutorul funcției accept(). Această funcție creează un nou socket dedicat pentru clientul respectiv, permițând serverului să gestioneze cererea clientului în mod izolat.
* Pentru a susține conexiuni multiple, serverul nostru va utiliza un thread pool. Fiecare fir de execuție (thread) va prelua cererea unui client și o va procesa independent de celelalte, astfel încât serverul să rămână receptiv și să nu blocheze alte cereri în timp ce răspunde la una.

1. ***Tratarea cererilor HTTP pe baza metodei (GET, POST, PUT, DELETE)***

* Fiecare cerere primită prin socket conține un mesaj HTTP, inclusiv metoda de cerere (GET, POST, PUT, DELETE) și URL-ul resursei solicitate.
* Serverul nostru va analiza mesajul și va utiliza metoda corespunzătoare pentru a răspunde.

1. ***Închiderea conexiunilor***

* După ce cererea a fost procesată și răspunsul a fost trimis clientului, serverul va închide socket-ul dedicat acelei conexiuni folosind funcția close(). Acest lucru eliberează resursele asociate și permite serverului să fie disponibil pentru noi conexiuni.
* Socket-ul principal, cel pe care serverul ascultă, va rămâne deschis până când serverul este oprit manual.

**Etapele Conexiunii între Client și Server**

1. ***Inițierea Cererii de Conexiune***

* Atunci când un client dorește să acceseze o pagină web de pe server, deschide un browser și introduce adresa IP a serverului și portul specificat (în acest caz, portul 80).
* Browserul trimite o cerere de conexiune către server prin protocolul TCP, iar această cerere ajunge la socket-ul serverului, care ascultă pe adresa și portul configurate.

1. ***Stabilirea Conexiunii TCP***

* Serverul primește cererea de conexiune și o acceptă, creând un nou socket dedicat pentru acea conexiune cu clientul respectiv.
* După acceptarea conexiunii, între server și client se stabilește o conexiune TCP, ceea ce asigură că datele vor fi transmise fiabil și în ordinea corectă.
* În această etapă, serverul este gata să primească cererea HTTP completă trimisă de client prin socket-ul deschis.

1. ***Transmiterea Cererii HTTP de către Client***

* Odată ce conexiunea TCP este stabilită, browserul trimite cererea HTTP, care conține metoda (GET, POST, PUT, DELETE), URL-ul resursei solicitate, și, în cazul cererilor POST sau PUT, datele atașate.
* Această cerere este transmisă prin socket și ajunge la server, care o analizează și decide asupra modului în care trebuie gestionată pe baza metodei HTTP utilizate.

1. ***Procesarea Cererii pe Server***

* Serverul analizează cererea primită, identifică metoda HTTP și resursa solicitată și execută operațiunile corespunzătoare. De exemplu:
* Pentru o cerere GET, serverul va localiza fișierul solicitat și îl va trimite clientului prin socket.
* Pentru o cerere POST, serverul va procesa datele trimise de client și va actualiza sau salva informațiile în funcție de cerință.
* PUT și DELETE vor permite modificarea sau ștergerea resurselor specificate, utilizând tot acest socket pentru transmiterea datelor între server și client.
* Dacă este necesar, serverul poate invoca interpretoare PHP sau JavaScript pentru a executa cod dinamic.
* După procesarea cererii, serverul generează un răspuns HTTP care include codul de stare (de exemplu, 200 OK pentru succes, 404 Not Found pentru resurse lipsă) și datele solicitate.

1. ***Trimiterea Răspunsului HTTP Către Client***

* Serverul trimite răspunsul HTTP către client prin socket-ul dedicat acelei conexiuni. Acest răspuns conține datele necesare pentru afișarea paginii web sau a resurselor solicitate, cum ar fi textul, imaginile, fișierele și alte elemente.
* În plus, răspunsul poate include și headere HTTP pentru a specifica tipul conținutului, lungimea acestuia și alte informații relevante pentru client.

1. ***Închiderea Conexiunii***

* După trimiterea răspunsului, serverul poate închide conexiunea, eliberând astfel resursele alocate socket-ului respectiv.
* În general, conexiunile sunt închise după fiecare cerere/ răspuns, dar în funcție de specificațiile proiectului, serverul poate permite conexiuni persistente (keep-alive) pentru a servi mai multe cereri succesive de la același client, fără a crea o nouă conexiune TCP pentru fiecare cerere.

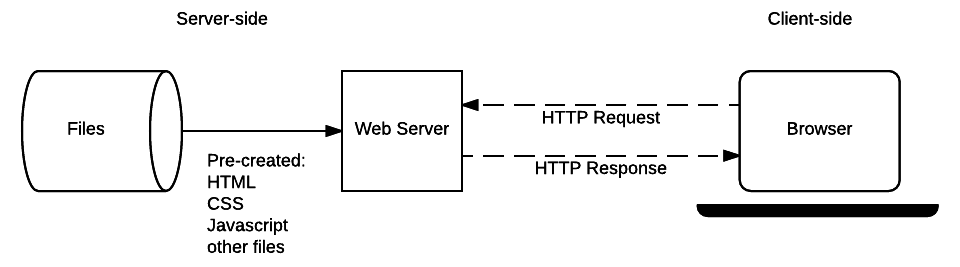


Image 1. Basic web server architecture for a static site. Sursa [1]

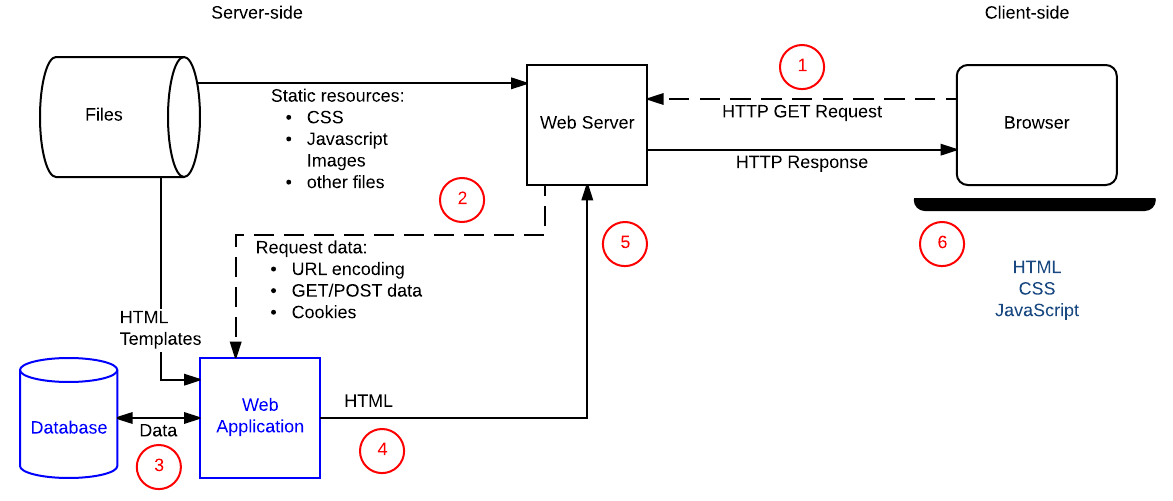


Image 2. Basic architecture for a dynamic website. Sursa [1]

# Descrierea serverului web implementat

TO BE CONTINUED

# Bibliografie

1. Introduction to the server side, [https://developer.mozilla.org/en-US/docs/Learn/Server-side/First\_steps/Introduction](https://en.wikipedia.org/wiki/Finite-state_machine)
2. How the web works: HTTP and CGI explained, [https://www.garshol.priv.no/download/text/http-tut.html](https://en.wikipedia.org/wiki/Finite-state_machine)