**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](#_Toc8499)

[2. Scopul proiectului 4](#_Toc24754)

[3. Descrierea HTTP și a metodelor de cerere 4](#_Toc25824)

[4. Conexiunea cu serverul 6](#_Toc26442)

[5. Descrierea serverului web implementat 7](#_Toc28252)

[5.1. Definiții și structuri 7](#_Toc15523)

[5.2. Variabile globale 8](#_Toc15365)

[5.3. Funcții 8](#_Toc297)

[5.4. Fluxul de execuție 8](#_Toc32706)

[Bibliografie 10](#_Toc12571)

**Listă de figuri**

[Image 1 . How Socket Works. Sursa [3] 6](#_Toc24964)

**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

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. |
| 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.

Folosind interfața de rețea cu socketuri, serverul nostru poate utiliza un set de funcții din pachetul standard C <sys/socket.h> și să-i permită serverului nostru să „vorbească” cu clienții prin internet. Acești clienți folosesc, de asemenea, socketuri pentru a comunica cu noi, deci un socket este practic un consens despre cum să comunicăm unii cu alții pe internet.

**Clienții trimit cereri și primesc răspunsuri de la servere, amândoi folosind socketuri.**

Pentru a începe, am creat și configurat un socket pentru server. Am configurat câteva lucruri:

* **AF\_INET**: folosesc IPv4 (în loc de IPv6)
* **SOCK\_STREAM**: folosesc TCP (în loc de UDP)
* **INADDR\_ANY**: serverul acceptă conexiuni din orice interfață de rețea

Funcția **bind()** leagă socketul de un port (în cazul nostru, portul 8080), ceea ce înseamnă că socketul va asculta pentru orice client care încearcă să se conecteze la portul 8080. Funcția **listen()** primește numărul maxim de conexiuni în așteptare (l-am setat la 10).

Acum, serverul nostru este configurat și gata să accepte conexiuni de la clienți.

# Descrierea serverului web implementat

## Definiții și structuri

* **#define PORT 8080:** Definește portul pe care serverul va asculta.
* **#define BUFFER\_SIZE 1048576:** Definește dimensiunea buffer-ului folosit pentru citirea fișierelor și trimiterea lor către client.
* **typedef struct { ... } Carte;:** Definește structura Carte, care conține informații despre o carte, precum titlu, autor, an de apariție, fișier PDF și imagine de copertă.

## Variabile globale

* **Carte carti[100];:** Un tablou pentru stocarea informațiilor despre cărțile încărcate din fișier.
* **int numar\_carti = 0;:** Contor pentru numărul de cărți încărcate.
* **pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;:** Un mutex pentru sincronizarea accesului la date partajate între thread-uri.

## Funcții

#### **get\_file\_extension:** Determină extensia fișierului, căutând ultima apariție a unui punct în numele fișierului.

#### **get\_mime\_type:** Returnează tipul MIME corespunzător extensiei fișierului (de exemplu, text/html, image/jpeg, application/pdf).

#### **load\_books:** Încarcă informațiile despre cărți din fișierul carti.txt (format CSV) și le stochează în tablou carti. Formatul fișierului este: titlu, autor, an, pdf\_file, image.

#### **is\_valid\_extension\_for\_download:** Verifică dacă o extensie de fișier este validă pentru a fi descărcată (imagine sau PDF).

#### **build\_http\_response:** Construiește un răspuns HTTP pentru a trimite un fișier către client. Include antetul Content-Type și, dacă fișierul este valid pentru descărcare, și antetul Content-Disposition. Dacă fișierul nu este găsit, trimite un răspuns de eroare 404.

#### **handle\_books\_page:** Construiește un răspuns HTTP care afișează o listă de cărți într-o pagină HTML. Fiecare carte este prezentată cu titlu, autor, an și două butoane pentru descărcarea imaginii de copertă și a fișierului PDF.

#### **handle\_client:** Funcția care gestionează cererile fiecărui client într-un thread separat. Analizează cererea HTTP și alege ce răspuns să trimită:

#### Dacă cererea este pentru GET /, trimite fișierul index.html.

1. Dacă cererea este pentru GET /books, trimite pagina cu lista de cărți.
2. Dacă cererea este pentru GET /download/, trimite fișierul cerut (coperte sau PDF-uri).

#### **start\_server:** Inițializează serverul:

1. Creează un socket TCP/IP.
2. Leagă socket-ul de adresa și portul specificate.
3. Ascultă conexiunile pe portul definit.
4. Acceptă conexiunile de la clienți și creează un thread separat pentru fiecare client folosind pthread\_create.

### ****funcția** main:**

1. Încarcă informațiile despre cărți din fișierul carti.txt folosind load\_books.
2. Pornește serverul prin apelarea funcției start\_server.

## Fluxul de execuție

1. Serverul începe să asculte pe portul 8080.
2. Când un client se conectează, un thread separat este creat pentru a gestiona cererea clientului.
3. Serverul răspunde la cereri HTTP:

* Dacă cererea este pentru GET /, trimite fișierul index.html.
* Dacă cererea este pentru GET /books, trimite lista de cărți.
* Dacă cererea este pentru GET /download/, trimite fișierul corespunzător (imagine sau PDF).

1. Serverul poate gestiona simultan mai multe conexiuni datorită utilizării thread-urilor.

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)
3. How I Built a Simple HTTP Server from Scratch using C, [https://dev.to/jeffreythecoder/how-i-built-a-simple-http-server-from-scratch-using-c-739](https://en.wikipedia.org/wiki/Finite-state_machine)