**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 3](#_Toc16801)

[2. Descrierea HTTP și a metodelor de cerere 4](#_Toc18035)

[3. Conexiunea cu serverul 5](#_Toc28427)

[4. Arhitectura sistemului 6](#_Toc9418)

[5. Funcţionalităţi 6](#_Toc14055)

[6. Implementarea serverului 6](#_Toc24757)

[7. Implementarea pool-ului de thread-uri 7](#_Toc5194)

[7.1. Structuri de date principale: 7](#_Toc12861)

[7.2. Funcționalități principale: 8](#_Toc2877)

[7.3. Detalii adiționale: 8](#_Toc4324)

[8. Tipuri MIME 9](#_Toc4716)

[Bibliografie 9](#_Toc18700)

**Listă de figuri**

[Figure 1 . Status Code 5](#_Toc8176)

**Listă de tabele**

[Table 1 . Metode de cerere 5](#_Toc1590)

[Table 2 . Tipuri de MIME 9](#_Toc7904)

# 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..

# Descrierea HTTP și a metodelor de cerere

**HTTP (Hypertext Transfer Protocol)** reprezintă un protocol la nivel aplicație care definește modalitatea de comunicatie dintre un server ce găzduiește un site cu pagini web și un client. Dintre serverele HTTP cele mai folosite sunt Apache, nginx și Microsoft IIS, iar dintre clientii HTTP Chrome, Mozilla Firefox și Opera.

Un server HTTP se mai numește și server web. Un client HTTP se mai numește și Browser.

Rolul unui server WEB este de a translata adrese numite URL-uri cerute de browser în fișiere și trimiterea conținutului acestora prin rețea (Internet sau LAN) la client. Protocolul HTTP folosește default portul TCP/80. Acesta este portul pe care serverul web ascultă conexiuni de la clienți.

Mesajele HTTP constau în cereri provenite de la client către server și răspunsuri provenite de la server către client.

HTTP-message = Request | Response

Orice mesaj HTTP este precedat de un HEADER care oferă informații cu privire la cererea sau răspunsul HTTP, sau cu privire la obiectul trimis în interiorul mesajului. Protocolul HTTP prezintă 4 tipuri de headere:

* General-header: cu aplicabilitate generală, atât pentru cereri, cât și pentru răspunsuri
* Request-header: cu aplicabilitate doar pentru cereri
* Response-header: cu aplicabilitate doar pentru răspunsuri
* Entity-header: definește meta-date cu privire la obiectul din corpul mesajului sau cu privire la resursa identificată prin cerere.

**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

Răspunsurile HTTP provin de regulă de la server și cuprind un element denumit Status Code (cod de stare). Status-Code este reprezentat de un număr întreg de trei cifre, unde prima cifră definește clasa de răspuns. Există 5 valori pe care le poate avea prima cifră din Status-Code:

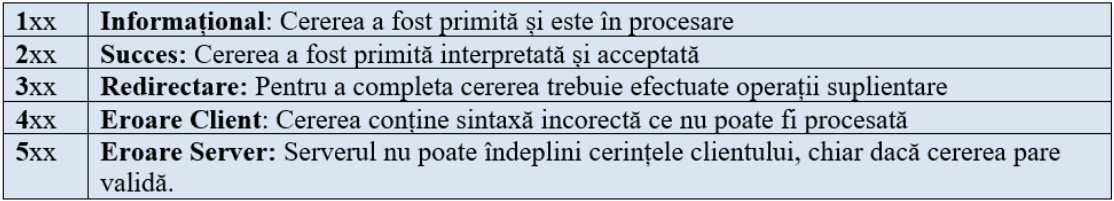


Figure 1. Status Code

# 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.

# Arhitectura sistemului

Acest document descrie implementarea unui server web care rulează pe portul 8080 și permite afișarea și descărcarea cărților dintr-o bibliotecă virtuală. Serverul poate gestiona mai multe conexiuni simultane folosind un thread pool.

Serverul este alcătuit din următoarele module principale:

* **threadpool.h și threadpool.c**: Implementarea unui pool de thread-uri care preia cereri de lucru dintr-o coadă.
* **server.c**: Serverul principal care acceptă conexiuni de la clienți și adaugă cererile primite în coada threadpool-ului.

# Funcţionalităţi

* **Listarea cărților**: Pagina /books afișează toate cărțile disponibile.
* **Descărcarea fișierelor**: PDF-urile și imaginile de copertă pot fi descărcate cu ajutorul butoanelor.
* **Feedback**: Formularul de feedback trimite informații către scriptul public/script.sh, care este executat folosind execvp.

# Implementarea serverului

#### Funcții de utilitate:

1. **get\_file\_extension:** Returnează extensia unui fișier dat, eliminând punctul (.) din față. Dacă fișierul nu are extensie, returnează un șir gol.
2. **get\_mime\_type:** Întoarce tipul MIME corespunzător unei extensii de fișier (ex. html -> text/html). Dacă extensia nu este recunoscută, returnează application/octet-stream.
3. **is\_valid\_extension\_for\_download:** Verifică dacă extensia unui fișier este validă pentru descărcare (png, jpg, jpeg, pdf).
4. **replace\_plus\_with\_space:** Înlocuiește toate caracterele + dintr-un șir cu spații. Util în procesarea datelor din formulare.
5. **extract\_form\_data:** Extrage valorile câmpurilor nume și prenume din corpul unei cereri POST HTTP. Returnează 1 dacă datele au fost extrase cu succes, altfel 0.
6. **run\_script\_feedback:** Rulează un script extern (public/script.sh) cu parametrii nume și prenume. Capturează ieșirea scriptului și o include în răspunsul HTTP.

#### Funcții de gestionare a cărților:

#### load\_books: Încarcă date despre cărți dintr-un fișier text (public/carti.txt) și le stochează într-un vector global carti. Fiecare carte este reprezentată de o structură carte.

1. **handle\_books\_page:** Generează o pagină HTML care listează toate cărțile încărcate. Include butoane pentru descărcarea copertei și a fișierului PDF al fiecărei cărți.

#### Funcții pentru construirea răspunsurilor HTTP:

1. **build\_http\_response:** Construiește un răspuns HTTP pentru un fișier specificat. Include antetul Content-Disposition dacă fișierul este valid pentru descărcare. Dacă fișierul nu există, trimite un răspuns 404.
2. **handle\_feedback\_submission:** Procesează cererile POST pentru feedback. Extrage datele din corpul cererii, rulează un script cu aceste date și trimite rezultatul scriptului înapoi clientului.

#### Funcții principale de gestionare a conexiunilor:

1. **handle\_client:** Gestionează o conexiune de la un client. Interpretează cererea HTTP și apelează funcțiile corespunzătoare (build\_http\_response, handle\_books\_page, etc.) în funcție de tipul cererii.
2. **start\_server:** Creează un server care ascultă pe portul specificat. Acceptă conexiuni de la clienți și le pasează unui thread pool pentru procesare concurentă.

#### Funcția principală:

1. **main:** Încarcă datele despre cărți folosind load\_books și pornește serverul cu start\_server.

# Implementarea pool-ului de thread-uri

### Structuri de date principale:

1. **tpool\_work\_t:** Reprezintă o sarcină (task) în coadă.

* func: Pointer către funcția ce urmează să fie executată.
* arg: Argumentul transmis funcției.
* next: Pointer către următoarea sarcină din coadă.

1. **tpool\_t:** Structura principală care reprezintă thread pool-ul.

* work\_first **și** work\_last: Primul și ultimul element din coada de sarcini.
* work\_mutex: Mutex pentru acces sincronizat la coadă.
* work\_cond: Condiție pentru notificarea firelor de execuție că au fost adăugate noi sarcini.
* working\_cond: Condiție utilizată pentru a semnala când toate sarcinile s-au terminat.
* working\_cnt: Numărul de sarcini în curs de procesare.
* thread\_cnt: Numărul total de fire de execuție active.
* stop: Flag pentru oprirea thread pool-ului.

### Funcționalități principale:

1. **Crearea unei sarcini (**tpool\_work\_create**)**

* Alocă dinamic memorie pentru o sarcină.
* Setează funcția și argumentul.

1. **Eliminarea unei sarcini (**tpool\_work\_destroy**)**

* Eliberează memoria alocată pentru o sarcină.

1. **Preluarea unei sarcini din coadă (**tpool\_work\_get**)**

* Returnează și elimină prima sarcină din coadă.

1. **Funcția de lucru a unui fir (**tpool\_worker**)**

* Preia și execută sarcini din coadă.
* Oprește execuția când stop devine true.

1. **Crearea thread pool-ului (**tpool\_create**)**

* Inițializează structura tpool\_t.
* Creează fire de execuție pentru procesarea sarcinilor.

1. **Adăugarea unei sarcini (**tpool\_add\_work**)**

* Adaugă o sarcină la coada de sarcini.
* Notifică firele de execuție despre o sarcină nouă.

1. **Așteptarea finalizării tuturor sarcinilor (**tpool\_wait**)**

* Blochează execuția până când toate sarcinile sunt procesate.

1. **Distrugerea thread pool-ului (**tpool\_destroy**)**

* Eliberează resursele asociate thread pool-ului.
* Așteaptă ca toate firele de execuție să termine.

### Detalii adiționale:

1. **Thread-uri detașate (**pthread\_detach**)**: Firele create sunt detașate pentru a evita necesitatea unui pthread\_join. Astfel, resursele lor sunt eliberate automat la terminare.
2. **Gestionarea concurenței**:

* Mutex-ul (work\_mutex) asigură acces sincronizat la structurile interne ale thread pool-ului.
* Condițiile (work\_cond, working\_cond) permit comunicarea între fire.

1. **Controlul opririi**:

* Flag-ul stop oprește thread pool-ul în mod controlat, eliberând toate resursele.

# Tipuri MIME

Tipurile MIME sunt importante pentru a indica browserului ce tip de conținut este transferat.

|  |  |
| --- | --- |
| **Extensie** | **Tip MIME** |
| .html/.htm | text/html |
| .txt | text/plain |
| .jpg/.jpeg | image/jpeg |
| .png | image/png |
| .pdf | application/pdf |

Table 2. Tipuri de MIME

# 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)
4. Thread Pool in C, <https://nachtimwald.com/2019/04/12/thread-pool-in-c/>
5. Analiza protocolului HHTP, [https://wiki.mta.ro/c/3/rc/lab/11/](https://nachtimwald.com/2019/04/12/thread-pool-in-c/)