**Mini server Web**

- cu suport pentru conexiuni multiple, cu thread-uri pooling -



Sumanariu Bianca

Popa Irina

C 113 C

Cuprins

[INTRODUCERE 3](#_Toc149893923)

[Scopul proiectului 3](#_Toc149893924)

[Lista de definitii 4](#_Toc149893925)

[Descrierea solutiei 4](#_Toc149893926)

[Mediu de dezvoltare 6](#_Toc149893927)

[IMPLEMENTARE 7](#_Toc149893928)

[Arhitectura 7](#_Toc149893929)

[Detalii implementare 8](#_Toc149893930)

[Flow-ul aplicatiei 9](#_Toc149893931)

[PLAN DE TESTARE 9](#_Toc149893932)

# INTRODUCERE

## Scopul proiectului

Mini server Web – cu suport pentru conexiuni multiple, cu thread-uri pooling are ca scop gestionarea cererilor HTTP de la clienti (solicitarea unei anumite resurse) si oferirea de raspunsuri rapide (informatii sau mesaje de eroare).



## Lista de definitii

Server = computer sau computer program care centralizeaza si manageriaza resurse sau servicii cerute de alte programe, aplicatii (clienti) printr-o conexiune

Socket = o legatura software dintre un client si un server in care datele sunt transmise bidirectional real-time

Thread pool = grup de thread-uri preinstantiate care sunt gata sa execute anumite task-uri date de procesor

HTTP (Hypertext Transfer Protocol) = protocolul cel mai des intalnit pentru accesarea si transmiterea informatiilor din/in Internet (www)

URL = identificator unic folosit pentru a localiza o resursa in Internet (adresa)

## Descrierea solutiei

Vom implementa o aplicatie care sa poata capta request-urile HTTP venite de la browsere(clienti) si sa raspunda la acestea.

Serverul va creea un pool de thread-uri prealocate, care pot prelua și procesa cererile.

Aplicatia primeste cererile HTTP pe port si le analizeaza(parseaza), obtinand informatii despre documentul/link-ul cerut.

Forma a unui string-request de la browser(client):

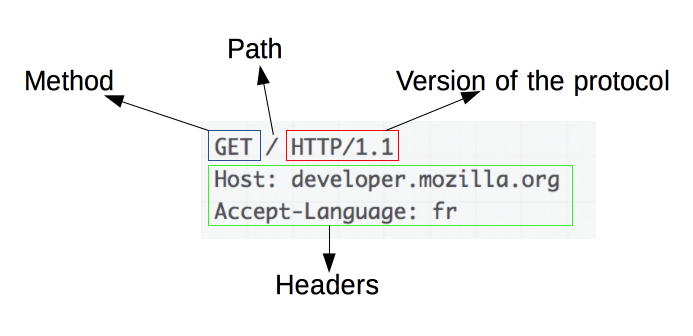
**"GET /request-URL HTTP/version"**, unde

GET = metoda HTTP (GET, POST, PUT, DELETE)

request-URL HTTP = URL-ul documentului cerut

version = versiunea HTTP-ului

**Exemplu: GET/pagina.html HTTP/1.1**



Serverul preia metoda din cererea HTTP care determină tipul de acțiune solicitată de client.

Serverul trebuie să trimită răspunsurile HTTP înapoi la clienți,

generandu-le corespunzator cererilor primite. In acest sens, exista o forma particulara de raspuns:

**HTTP/[VER] [CODE] [TEXT]**

**Field1: Value1**

**Field2: Value2**

**...Document content here... <------ (daca nu este link)** , unde

VER = versiunea protocolului HTTP

CODE = status (success sau nu)

Aceste coduri se impart in 5 subgrupuri:

#### 1xx: Informational (experimental purposes only)

#### 2xx: Successful

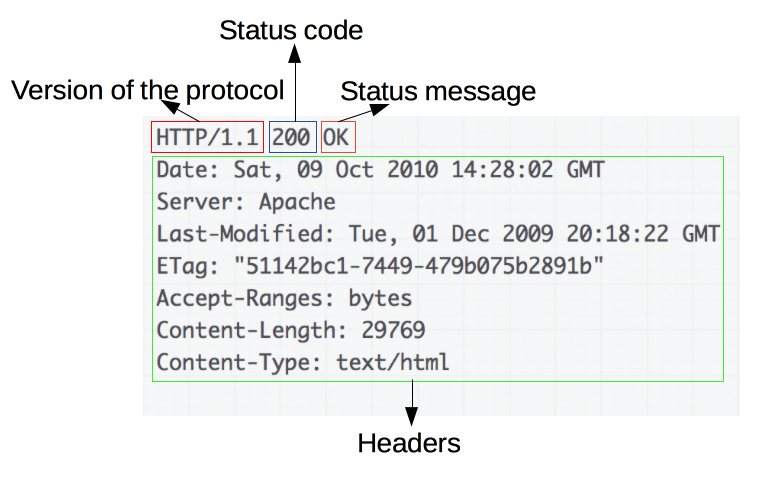
#### 3xx: Redirection (inseamna ca resursa ceruta este in alta parte, iar clientul trebuie sa incerce la o noua adresa)

#### 4xx: Client error (404:Not Found - cel mai “placut” si des mesaj)

#### 5xx: Server error

TEXT = text human-readable care sa confirme CODE

Field1, Field2 = metadate despre content-ul cerut (data crearii, numele server-ului, content-length etc)



## Mediu de dezvoltare

Programul dezvoltat va putea fi rulat pe o masinarie ce foloseste ca sistem de operare Linux, datorita functionalitatilor prezentate pe acest tip de masinarii (VM).

# IMPLEMENTARE

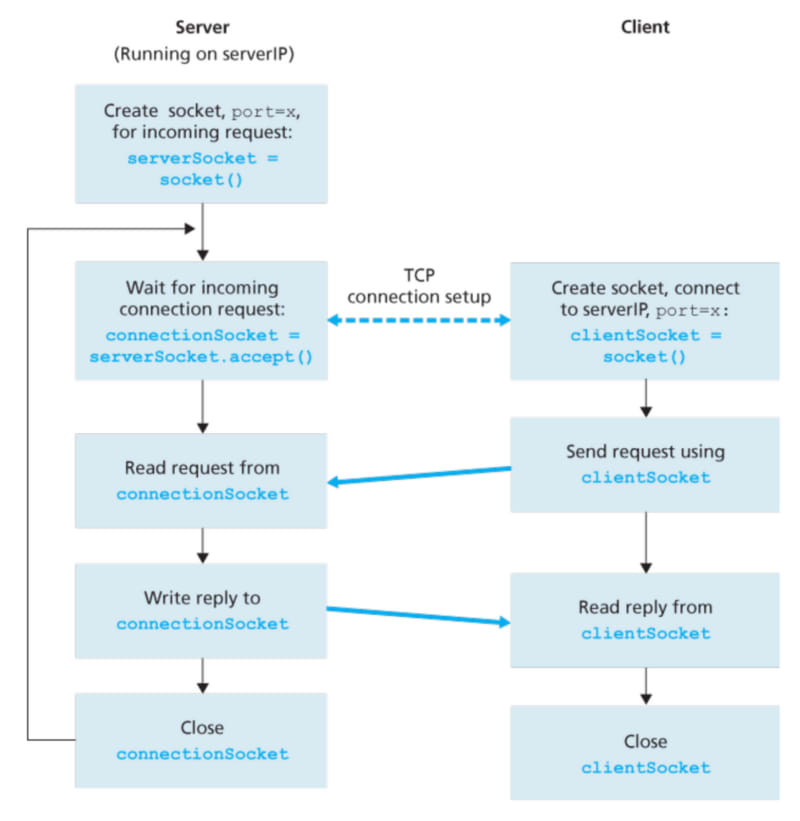
## Arhitectura

Arhitectura aplicatiei va fi bazata pe idea de fork si de thread pooling (pentru optimizare).

## Detalii implementare

Pasii urmati pentru crearea aplicatiei:

1. Crearea socket-ului de ascultare
2. Legarea socket-ului la o adresa IP si la un port (vor fi luate dintr-un fisier de configurare) si asteptarea conexiunilor folosind bucle infinite.



1. Serverul va avea doua moduri (optiuni) :

Modul Fork: pentru fiecare cerere se creează un proces copil pentru a gestiona cererea

Modul Threads: unde se folosește un pool de fire de execuție pentru a gestiona cererile concurente (pool-ul de thread-uri așteaptă cereri de la clienți, când o cerere este primită este plasată într-o coadă)

E nevoie de gestionarea conexiunii in functie de aceste moduri.

1. Crearea de functii pentru procesarea cererilor .
2. Generarea raspunsului (transforma pagina web sub forma de sir de caractere) si trimiterea acestuia.
3. Inchiderea conexiunii.

## Flow-ul aplicatiei

Initializarea serverului 🡪 Configurarea serverului 🡪 Asteptarea conexiunii 🡪 Gestionarea conexiunii 🡪 Citirea cererii HTTP 🡪 Procesarea cererii 🡪 Generarea raspunsului 🡪 Trimiterea raspunsului 🡪 Inchiderea conexiunii 🡪 Continuarea buclei

# PLAN DE TESTARE

1. **Testarea funcționării de bază**

Verificarea dacă serverul poate fi pornit și oprit fără erori.

Accesarea serverului prin intermediul unui browser sau unui client HTTP pentru a verifica dacă poate servi pagini statice.

1. **Testarea răspunsului la cereri HTTP:**

Trimiterea cererilor HTTP GET pentru diferite resurse (HTML, imagini, fișiere etc) și verificarea dacă serverul le livrează corect.

Verificarea codurilor de stare HTTP returnate de server pentru cereri corecte și eronate (de exemplu, 404 pentru resurse inexistente).

1. **Testarea funcționalității de multithread:**

Verificarea creării și gestionării thread-urilor copil, pentru fiecare conexiune aparuta.

Trimiterea cererilor HTTP și verificarea dacă serverul poate gestiona mai multe cereri prin bifurcare.

1. **Testarea funcționalității de thread pooling:**

Trimiterea unui număr mare de cereri simultane și monitorizarea comportamentului serverului pentru a se asigura că thread pooling funcționează corect.

# Bibliogarfie

<https://copyprogramming.com/howto/thread-pool-in-c-for-web-server>

<https://dev.to/jeffreythecoder/how-i-built-a-simple-http-server-from-scratch-using-c-739>

<https://github.com/JeffreytheCoder/Simple-HTTP-Server>

<https://www.w3schools.com/tags/ref_httpmethods.asp>