Progetto di Reti

Barzanti Alan **0001116797**
Mariani Federico **0001116947**
Mondardini Marco **0001114901**

Giugno 2025

Contents

1 Introduzione

Il progetto realizzato è un semplice server web in Python che utilizza le socket per gestire le richieste HTTP.
L'obiettivo è fornire file statici contenuti in una directory definita (`WEB_ROOT`) in risposta a richieste HTTP, in particolare il metodo GET. Se il file non viene trovato, viene restituito un errore 404.

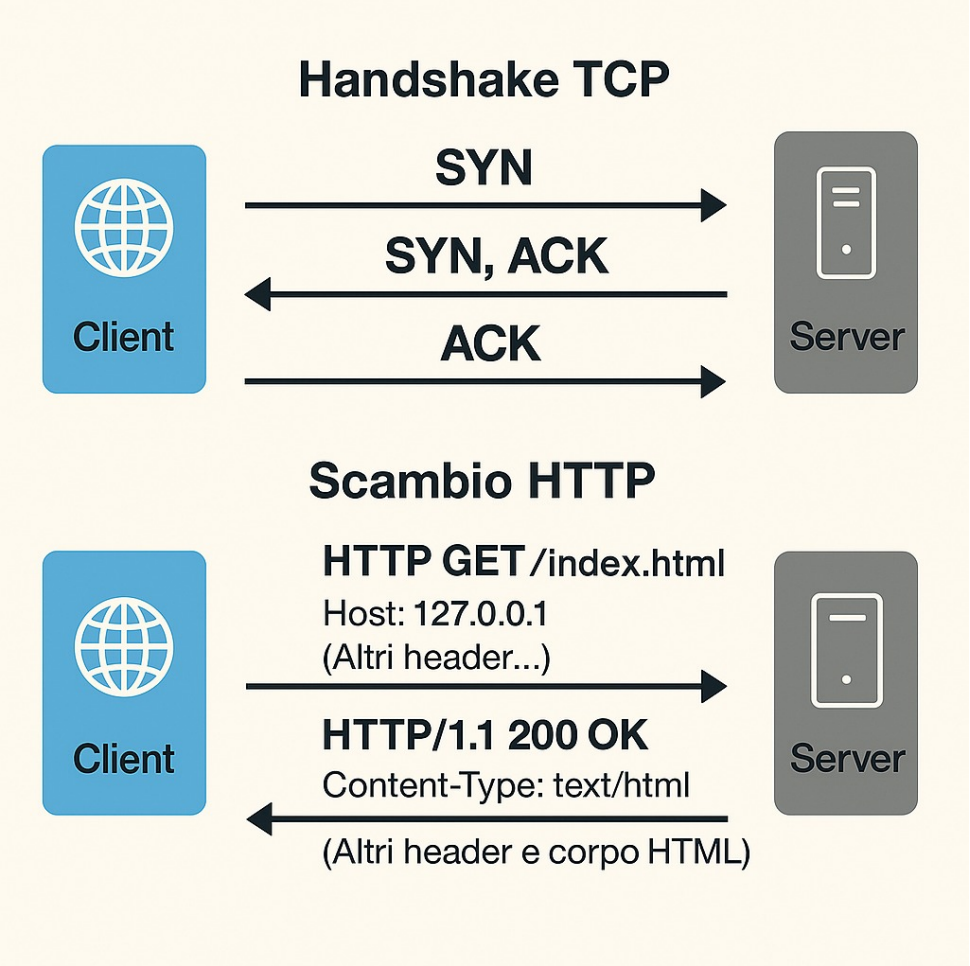


Figure 1: Schema di cosa ci si aspetta.

2 Configurazione del Server e Contesto

- **HOST:** 127.0.0.1 (server accessibile in locale)
- **PORT:** 8080 (porta di ascolto in ambiente di sviluppo)
- **WEB_ROOT:** `www` (directory da cui vengono serviti i file)
- **MIME_TYPES:** Dizionario che definisce i tipi MIME per le estensioni comuni (.html, .css, .jpg, .png, .ico)

3 Analisi delle Funzioni

3.1 get_mime_type(file_path)

Questa funzione estrae l'estensione del file utilizzando `os.path.splitext()` e restituisce il relativo MIME type basato su un dizionario predefinito. Se l'estensione non è presente, viene usato `application/octet-stream`.

3.2 log_request(method, path, status)

Registra in console ogni richiesta HTTP nel seguente formato:

`[YYYY-MM-DD HH:MM:SS] <METODO> <PATH> -> <STATUS>`

Questo aiuta a monitorare le richieste processate dal server.

3.3 handle_request(client_socket)

- Riceve la richiesta HTTP (fino a 1024 byte) e la divide in righe.
- Parsea la prima riga per estrarre il metodo e il path richiesto.
- Se il metodo non è GET, invia una risposta HTTP 405 (Method Not Allowed) e chiude la connessione.
- Se il path richiesto è "/", viene reindirizzato a "/index.html".
- Costruisce il percorso completo concatenando `WEB_ROOT` e il path (senza il carattere iniziale "/").
- Se il file esiste:
  - Legge il file in modalità binaria.
  - Determina il MIME type tramite `get_mime_type`.
  - Risponde con HTTP 200 OK includendo il contenuto del file e l'header `Content-Type` appropriato.
- Se il file non esiste:
  - Invia una risposta HTTP 404 Not Found con un messaggio HTML.
- Dopo aver inviato la risposta, registra l'operazione tramite `log_request` e chiude la connessione.

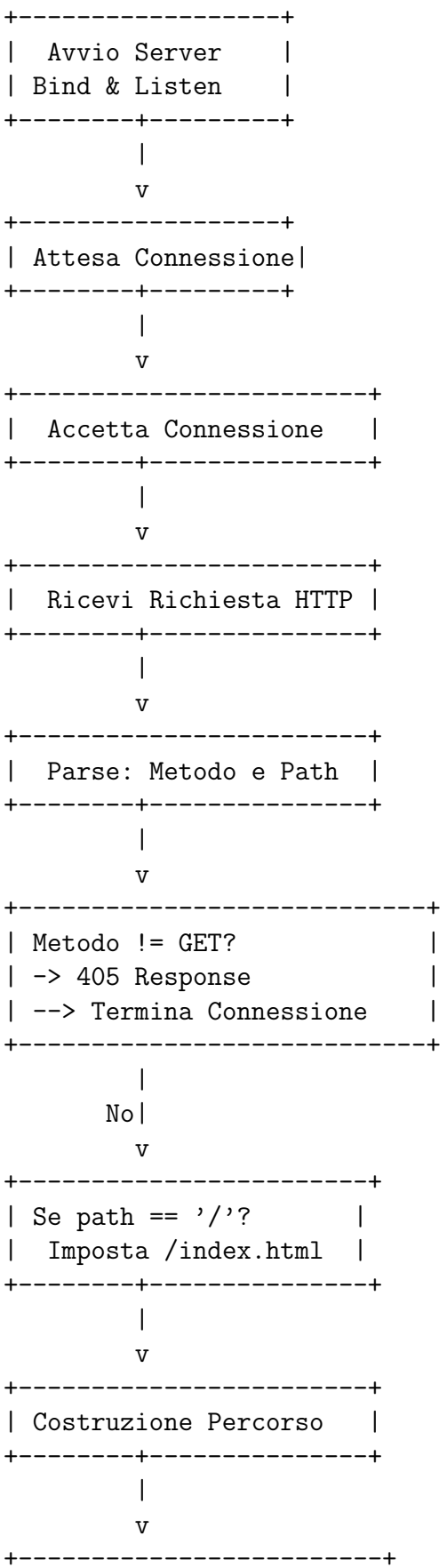
3.4 run_server()

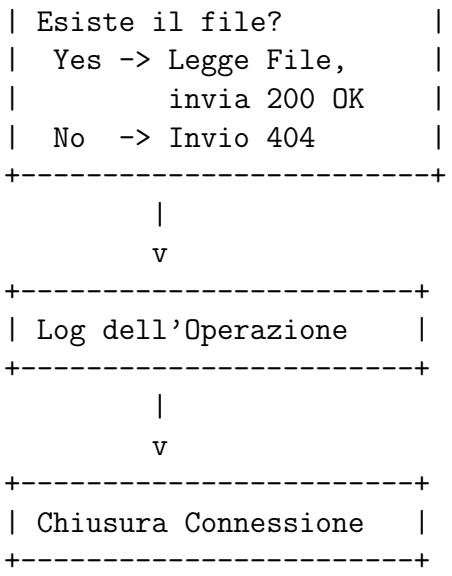
- Crea un socket TCP e lo associa all'HOST e alla PORT definiti.
- Imposta il socket in modalità ascolto per connessioni in coda (fino a 5).
- Entra in un ciclo infinito in cui:
  - Accetta una connessione in arrivo.
  - Gestisce la richiesta chiamando `handle_request`.

3.5 Blocco Principale

Il costrutto `if __name__ == '__main__'` garantisce che il server si avvii solo se lo script viene eseguito direttamente, e non durante eventuali importazioni come modulo in altri script.

4 Diagramma di Flusso




| Esiste il file?
| Yes -> Legge File,
| invia 200 OK
| No -> Invio 404
| Log dell'Operazione
| Chiusura Connessione

5 Approfondimenti e Possibili Miglioramenti

5.1 Gestione dei Metodi HTTP

Il server attualmente supporta solo il metodo GET. Potrebbe essere esteso per gestire metodi come POST o HEAD per ulteriori funzionalità.

5.2 Concorrenza e Scalabilità

Essendo un server sincrono, gestisce una richiesta per volta. Al miglioramento si potrebbe ricorrere a:

- Multithreading o multiprocessing.
- Tecnologie asincrone per gestire connessioni multiple in modo più efficiente.

5.3 Sicurezza e Robustezza

- Una validazione più rigorosa del path richiesto è fondamentale per prevenire attacchi come il *directory traversal*, cioè l'accesso a file e directory al di fuori della root.
- Gestire le eccezioni con blocchi `try-except` attorno alle operazioni di I/O renderebbe il server più stabile.

5.4 Logging Avanzato

Implementare un sistema di logging più avanzato, ad esempio scrivendo il log su file o integrando sistemi di monitoraggio, può essere utile per analisi e debug.

6 Conclusioni

Il progetto *Webserver Project* è un esempio didattico che illustra i principi fondamentali di un server HTTP scritto in Python. Fornisce una base solida per comprendere come avviene la comunicazione tra client e server, pur rimanendo semplice ed estensibile.

7 Informazioni Aggiuntive

- **Testing:** Puoi verificare il funzionamento del server utilizzando un browser (http://127.0.0.1:8080)
- **Documentazione:** Una documentazione dettagliata e commenti esplicativi facilitano la manutenibilità e la collaborazione su progetti più complessi.
- **Espansioni Future:** Si consiglia di considerare l'aggiunta di supporto per ulteriori metodi HTTP, la gestione della concorrenza e misure di sicurezza avanzate.