Elaborato per il corso
Programmazione di Reti

Grazia Bochdanovits de Kavna

1 settembre 2025

Indice

Capitolo 1

Introduzione

Il presente progetto ha l'obiettivo di realizzare un **web server HTTP** in **Python**, utilizzando socket per gestire le connessioni di rete. Il server è progettato per rispondere sulla **porta 8080** di `localhost` e servire contenuti statici dalla directory `www/`.

Il lavoro implementa tutti i requisiti minimi richiesti dalla traccia: gestione delle richieste `GET` con risposta 200, servizio di almeno tre pagine HTML statiche, e corretta gestione degli errori 404 per file inesistenti.

Sono state inoltre implementate le estensioni opzionali previste, tra cui il supporto per *MIME types* (HTML, CSS), il logging delle richieste con *timestamp*, e un layout responsive con animazioni CSS.

Il server utilizza **multithreading** per gestire connessioni concorrenti e include misure di sicurezza per prevenire attacchi di directory traversal. Il progetto comprende anche pagine di errore personalizzate (`404.html`, `501.html`) e un design moderno.

Capitolo 2

Architettura del progetto

L'architettura del progetto si basa su una chiara separazione tra la logica applicativa del server e i contenuti statici destinati al client. Il sistema è organizzato nei seguenti componenti principali:

- **server.py**: implementa il web server HTTP multithread, in ascolto sulla porta 8080, responsabile della gestione delle connessioni e del servizio dei file statici.
- **www/**: directory contenente il sito web statico, composto da cinque pagine HTML e un file CSS condiviso, con layout responsive e design moderno.

Capitolo 3

Implementazione del web server

3.1 Struttura generale

Il web server è stato implementato in Python utilizzando esclusivamente moduli della standard library. L'architettura segue il pattern classico dei server HTTP multi-threading, dove ogni connessione client viene gestita da un thread separato. La classe principale `HTTPServer` incapsula tutta la logica del server.

La struttura del codice è organizzata nei seguenti componenti principali:

- **Classe HTTPServer**: Classe principale che gestisce l'intero ciclo di vita del server
- **Gestione delle connessioni**: Accettazione delle connessioni e creazione di thread dedicati
- **Parsing delle richieste**: Analisi degli header HTTP e estrazione dei parametri
- **Resolving dei path**: Conversione sicura degli URL in percorsi del filesystem
- **Servizio dei file**: Gestione dei file statici con controllo dei permessi
- **Gestione degli errori**: Produzione di pagine di errore appropriate

3.2 Gestione delle richieste

Il processo di gestione delle richieste segue un flusso ben definito:

3.2.1 Accettazione delle connessioni

Il ciclo principale del server, gestito dal metodo `_run_server_loop` della classe `HTTPServer`, rimane in attesa su una socket configurata con l'opzione di riutilizzo dell'indirizzo (`SO_REUSEADDR`). Utilizzando un timeout sulla socket, il ciclo può essere interrotto in modo pulito. Quando una nuova connessione viene accettata, per ogni client viene creato un nuovo thread dedicato (utilizzando il modulo `threading`) che invoca il metodo `_handle_client`, il quale si occuperà di gestire tutta la comunicazione con quel client specifico. La gestione delle eccezioni assicura che errori di accettazione o l'interruzione da terminale (tramite `KeyboardInterrupt`) vengano gestiti correttamente, portando a una chiusura ordinata del server.

3.2.2 Parsing della richiesta

Il metodo `_process_request` analizza la richiesta HTTP ricevuta, estraendo il metodo, il path e la versione del protocollo dalla prima linea della richiesta. Il parser verifica che la richiesta sia ben formata e contenga esattamente tre componenti. Sono supportate esclusivamente richieste `GET` secondo lo standard HTTP/1.x. Richieste con metodi diversi vengono rifiutate con un errore `405 Method Not Allowed`, mentre versioni del protocollo non supportate generano un errore `400 Bad Request`. Se il parsing ha successo, il controllo passa al metodo `_handle_get` per la gestione specifica della richiesta GET.

3.2.3 Resolving del percorso

Il metodo `_resolve_path` si occupa di garantire sicurezza durante la conversione degli URL in percorsi filesystem. Il percorso richiesto viene prima decodificato (`unquote`) e analizzato (`urlparse`) per estrarre il pathname, quindi normalizzato (`os.path.normpath`) per rimuovere riferimenti relativi ridondanti. Il path viene poi ancorato alla directory root dei contenuti statici (`www_dir`). La misura di sicurezza fondamentale è il controllo che il percorso assoluto risultante inizi effettivamente con il percorso di `www_dir`, prevenendo così attacchi di directory traversal che potrebbero tentare di accedere a file al di fuori della directory consentita. In caso di errore o di violazione di questo controllo, il metodo restituisce `None`, inducendo il server a rispondere con un errore `403 Forbidden`.

3.2.4 Servizio dei file

I file vengono serviti con il corretto Content-Type determinato automaticamente dal metodo `_serve_file`. Sono implementati diversi controlli di sicurezza e validazione:

- **Dimensioni massime**: I file superiori a 10MB vengono rifiutati con errore `413 Request Entity Too Large`
- **Tipi di file supportati**: File con estensioni dinamiche (`.php`, `.jsp`, `.asp`) vengono deliberatamente rifiutati con errore `501 Not Implemented`
- **Gestione automatica delle directory**: Se il path richiesto corrisponde a una directory, viene automaticamente servito il file `index.html` in essa contenuto, se presente
- **Rilevamento MIME type**: Il tipo MIME viene determinato automaticamente usando il modulo `mimetypes`, con fallback su `application/octet-stream`

La gestione delle eccezioni assicura che errori comuni come file non trovati o permessi insufficienti vengano convertiti negli appropriati codici di stato HTTP (`404 Not Found` e `403 Forbidden` rispettivamente).

3.2.5 Gestione degli errori

Il server produce pagine di errore appropriate per i vari codici di stato HTTP attraverso il metodo `_get_error_content`. Il sistema cerca prima pagine di errore personalizzate nella directory `www` (con naming convention `[codice].html`). Se non trova una pagina personalizzata per quel specifico codice di errore, genera dinamicamente una pagina di default con un messaggio di errore chiaro e un link per tornare alla homepage. Questo approccio ibrido offre flessibilità nella personalizzazione mantenendo al contempo una risposta appropriata per tutti i possibili errori.

3.2.6 Logging

Tutte le attività del server vengono registrate attraverso il modulo `logging` di Python. Il sistema è configurato per scrivere contemporaneamente su file (`server.log`) e su console, includendo per ogni entry un timestamp, il livello di gravità, e il messaggio descrittivo. Ogni richiesta viene loggata con l'indirizzo IP del client e la prima linea della richiesta HTTP, fornendo un audit trail completo delle attività del server mentre mantiene una dimensione gestibile dei log.

Capitolo 4

Sito web statico

Il sito web è composto da cinque pagine HTML statiche, tutte stilizzate mediante un unico file CSS condiviso (`common.css`).

4.1 Pagine HTML

- **index.html**: Homepage del sito con navigazione principale
- **login.html**: Pagina di autenticazione con form per le credenziali
- **contact.html**: Pagina di contatto con informazioni e form
- **404.html**: Pagina di errore personalizzata per risorse non trovate
- **501.html**: Pagina per funzionalità non ancora implementate

Capitolo 5

Conclusioni

Il server sviluppato in Python soddisfa pienamente i requisiti della traccia, rispondendo su `localhost:8080`, gestendo correttamente richieste GET e fornendo pagine di errore personalizzate. Sono state realizzate tutte le estensioni opzionali, includendo la gestione dei MIME type, il logging delle richieste e un sito statico con layout responsive.

5.1 Sviluppi Futuri

Il progetto, pur essendo completo e funzionale, può essere esteso con diverse funzionalità avanzate come il supporto HTTPS per comunicazioni cifrate e l'implementazione di metodi HTTP aggiuntivi come POST per gestire l'invio di dati dai form. Queste estensioni renderebbero il server più versatile e adatto ad applicazioni reali.

Appendice A

Guida utente

A.1 Installazione e Avvio

1. **Clonazione del repository**:
   `git clone https://github.com/this-Grace/ProgrammazioneReti-project.git`
2. **Accesso alla directory**:
   `cd ProgrammazioneReti-project`
3. **Avvio del server**:
   `python server.py`
4. **Accesso al sito**:
   Aprire un browser e navigare su `http://localhost:8080`

A.2 Accesso alle pagine

- **Homepage**: `http://localhost:8080/`
- **Pagina di login**: `http://localhost:8080/login.html`
- **Pagina di contatti**: `http://localhost:8080/contact.html`
- **Pagina 404**: `http://localhost:8080/404.html`
- **Pagina 501**: `http://localhost:8080/501.html`

A.2.1 Note sull'accesso

- La root `/` restituisce automaticamente `index.html`.

- File non esistenti mostrano la pagina `404.html`.
- Il form di login reindirizza a `501.html`.

A.3 Arresto del Server

- Premere `Ctrl+C` nel terminale.
- Attendere la chiusura delle connessioni.
- Verificare il messaggio: `Server stopped.`