Progetto Programmazione di Reti (Traccia 2)

Matteo Bambini

Luglio 2021

1 Scelte di Progetto

Il progetto è diviso in due file sorgenti: `app.py` e `httphandler.py`.

Il file `app.py` contiene il codice per avviare l'applicazione e per istanziare e gestire il web server. Contiene inoltre il codice per fare il parsing degli argomenti passati da riga di comando che permettono di personalizzare la porta su cui far ascoltare il server e l'indirizzo IP su cui fare il bind e l'handler che cattura i segnali di interruzione da tastiera e interrompe il server in modo da liberare la porta in uso.

Il file `httphandler.py` contiene la classe HttpHandler, ovvero la classe che serve per gestire le richieste HTTP e che estende `http.server.SimpleHTTPRequestHandler`. Ho deciso di realizzare una sotto-classe perché gestisco il path in modo dinamico, ovvero i path non sono cartelle e file come nelle situazioni più classiche, ma vengono gestiti dal codice che invia pagine create dinamicamente. Oltre al metodo `do_GET`, che è un estensione di quello presente nella classe padre, ho deciso di inserire i seguenti metodi:

- `__set_response`: crea la risposta e invia gli header necessari prima della pagina vera e propria
- `__load_services`: carica la lista dei servizi dal file services.json
- `__index`: renderizza la pagina principale partendo dal file public/index.html
- `__service`: renderizza la pagina di un determinato servizio partendo dal file public/service.html
- `__file_pdf`: restituisce il file PDF

2 Strutture Dati

L'unica struttura dati che ho utilizzato è il JSON. L'ho utilizzato per memorizzare le informazioni riguardanti i servizi all'interno del file services.json in modo da poterle caricare e generare dinamicamente le pagine che necessitano delle informazioni di uno o più servizi.

3 Thread

La gestione dei thread è eseguita in modo molto semplice: tutte le volte che viene stabilita una nuova connessione TCP viene avviato un nuovo thread per gestire la ricezione e l'invio dei dati. Questa operazione è eseguita in automatico dalla classe `socketserver.ThreadingTCPServer`. Quando viene inviato il segnale di chiusura il programma non aspetta che ogni thread completi la propria esecuzione, ma si chiude immediatamente e i thread ancora attivi finiranno il loro ciclo di vita normalmente. Questo avviene grazie all'opzione `daemon_threads` di `socketserver.ThreadingTCPServer` che permette appunto di creare i suddetti daemon thread.