Progetto di Programmazione di Reti

Sara Panfini, Matricola: 0001082217

July 2, 2024

Contents

1 Introduzione

Il progetto ha come obiettivo l'implementazione di un web server semplice in Python in grado di gestire richieste HTTP GET e di servire file statici. Questo documento descrive la struttura del codice, il suo funzionamento e le modalità di utilizzo.

2 Struttura del Codice

- Importazione dei Moduli
- Lettura degli Argomenti da Riga di Comando
- Inizializzazione del Server
- Gestione dei Segnali
- Avvio del Server

3 Descrizione del Codice

3.1 Importazione dei Moduli

Vengono prima importati i moduli necessari.

```
3   import sys, signal
4   import http.server
5   import socketserver
```

- **sys**, utilizzato per leggere gli argomenti da riga di comando
- **signal**, utilizzato per gestire i segnali
- **http.server**, utilizzato per gestire le richieste HTTP
- **socketserver**, utilizzato per gestire connessioni su socket

3.2 Lettura degli Argomenti da Riga di Comando

Questo frammento di codice gestisce gli argomenti passati da riga di comando.

```
7   if sys.argv[1:]:
8       ip_address = sys.argv[1]
9   else:
10      ip_address = '127.0.0.1'
11
12  if sys.argv[2:]:
13      port = int(sys.argv[2])
14  else:
15      port = 8080
```

Se forniti, gli argomenti della riga di comando vengono utilizzati per configurare l'indirizzo IP e la porta del server. Altrimenti, sono utilizzati i valori predefiniti *'127.0.0.1'* (localhost) per il server e *'8080'* per la porta.

3.3 Inizializzazione del Server

In questo blocco di codice viene istanziato e configurato il server.

```
17    server = socketserver.ThreadingTCPServer((ip_address, port),
18        http.server.SimpleHTTPRequestHandler)
19
20    server.daemon_threads = True
21
22    server.allow_reuse_address = True
```

Viene usata la funzione *'ThreadingTCPServer'*, che è in grado di gestire più richieste simultaneamente, avviando un nuovo thread per ogni richiesta. La classe *'http.server.SimpleHTTPRequestHandler'* fornisce i file dalla directory corrente e dalle sottodirectory. L'attributo *'server.daemon_threads'*, se impostato a "True", permette di gestire i thread in modo autonomo e, quindi, di terminare il server principale senza attendere che tutti i thread siano completati. Inoltre, l'attributo *'allow_reuse_address'*, con il flag settato a "True", consente al server di riutilizzare la stessa porta, senza dover attendere che il Kernel rilasci la porta sottostante.

3.4 Gestione dei Segnali

Questa sezione di codice definisce un gestore di segnali per catturare l'interruzione del programma.

```
23    def signal_handler(signal, frame):
24        print('Exiting http server (Ctrl + C pressed)')
25        try:
26            if(server):
27                server.server_close()
28        finally:
29            sys.exit(0)
30
31    signal.signal(signal.SIGINT, signal_handler)
```

La funzione *signal.signal()'* associa il segnale *'signal.SIGINT'*, che interrompe l'esecuzione se viene digitata la sequenza "Ctrl + C", alla funzione *'signal_handler()'*, definendo un handler personalizzato. Quando viene premuta la combinazione di tasti "Ctrl + C", quindi, il server viene chiuso utilizzando *'server.server_close()'* per rilasciare tutte le risorse allocate e poi il programma termina con *'sys.exit(0)'*.

3.5 Avvio del Server

In questo blocco di codice viene avviato il server.

```
33  try:
34      while True:
35          print('Server HTTP')
36          print('Indirizzo IP:', ip_address)
37          print('Port:', port)
38          print('Per terminare premere Ctrl + C')
39          server.serve_forever()
40  except KeyboardInterrupt:
41      pass
42
43  server.server_close()
```

Il ciclo "while" mantiene il server in esecuzione in un ciclo infinito e la funzione *'server.serve_forever()'* gestisce le richieste HTTP fino a quando il server non termina o viene interrotto. Se viene segnalata l'eccezione *'KeyboardInterrupt'*, causata dall'interruzione da tastiera, si esce dal ciclo ed il server viene chiuso.

4 Esecuzione del Codice

Per eseguire il codice da riga di comando è necessario aprire un terminale ed accedere alla directory contenente i file. Il comando può essere eseguito passando degli argomenti, se si preferisce specificare un indirizzo IP ed una porta, o senza argomenti, se si vogliono usare i valori predefiniti.

- **Senza Argomenti**

```
>python webserver.py
```

- **Con Argomenti**

```
>python webserver.py 127.0.0.1 9090
```

Dopo l'esecuzione del comando, a terminale si potrà vedere l'indirizzo del server.

```
Server HTTP all'indirizzo http://127.0.0.1:8080
Indirizzo IP: 127.0.0.1
Port: 8080
Per terminare premere Ctrl + C
Chiusura server HTTP (Ctrl + C)
```