Università di Bologna - Campus di Cesena
Ingegneria e Scienze Informatiche (8615)

"Programmazione di reti"

Traccia 2: Web Server

Matricola: 0001080182

Manuele D'Ambrosio

Anno Accademico 2023 - 2024

Introduzione

Traccia 2: Web Server Semplice

Creare un web server semplice in Python che possa servire file statici (come HTML, CSS, immagini) e gestire richieste HTTP GET di base. Il server deve essere in grado di gestire più richieste simultaneamente e restituire risposte appropriate ai client.

Descrizione

Il web server è stato realizzato in Python ed è in grado di gestire più richieste HTTP GET simultaneamente mediante il multithreading. Nello specifico il server serve le richieste mostrando una semplice pagina web scritta in HTML.

Esecuzione

Per eseguire il server è sufficiente posizionarsi col terminale nella directory contenente il file "MyWebServer.py" ed eseguirlo tramite il comando:
$ python MyWebServer.py
Dopo averlo fatto apparirà a terminale la scritta "Server http://localhost:8080" dopodiché sarà possibile clickare sul link oppure copiarlo nella barra degli indirizzi di un browser per poter visionare la pagina restituita dal server. Di default il server si mette in ascolto sulla porta 8080 tuttavia è possibile passare un differente numero di porta da riga di comando Se si vuole chiudere il server bisognerà premere la combinazione di tasti 'CTRL+C'.

Codice Python

Package utilizzati

```
1 import sys, signal
2 import http.server as hs
3 import socketserver as ss
```

- Il modulo *sys* mette a disposizione delle funzionalità di base, in questo caso è stato utilizzato per ricevere eventuali argomenti da linea di comando e anche per effettuare una chiusura corretta del server.
- Il modulo *signal* permette di gestire i segnali come ad esempio quello della combinazione CTRL+C che consente di chiudere il server.
- Il modulo *http.server* (rinominato hs) fornisce degli handler per le richieste http.
- Il modulo *socketserver* (rinominato ss) semplifica il compito di scrivere server di rete.

Creazione del server address

```
1 if sys.argv[1:]:
2   PORT = int(sys.argv[1])
3 else:
4   PORT = 8080
5 HOST = 'localhost'
6 server_address = (HOST, PORT)
```

Il numero di porta '**PORT**' di default viene impostato a **8080** la quale è spesso utilizzata da server Web o altre applicazioni che utilizzano **HTTP** o altri protocolli **TCP**, è tuttavia possibile cambiarlo passando un nuovo numero di porta da linea di comando al momento dell'esecuzione del server. L'indirizzo IP del server '**HOST**' viene impostato a ***'localhost'*** che equivale all'indirizzo **127.0.0.1** il quale idenfica la stessa macchina da cui è stato lanciato lo script. Dopodiché viene assegnata la coppia di valori alla variabile '**server_address**' che verrà utilizzata successivamente.

Gestione simultanea di più richieste

```
handler = hs.SimpleHTTPRequestHandler
server = ss.ThreadingTCPServer(server_address, handler)
```

Viene creato un server **TCP** tramite il server address che è stato precedentemente definito e un **handler** che si occupa della gestione delle richieste del server. La classe **ThreadingTCPServer** viene utilizzata per creare un nuovo thread per ogni richiesta ricevuta, in questo modo possono essere gestiti numerosi **client** simultaneamnente.

Impostazioni del server

```
server.daemon_threads = True
```

```
server.allow_reuse_address = True
```

- **daemon_threads**: I thread creati dal server vengono impostati come *daemon thread* e in quanto tali non impediscono al server di chiudersi anche se non sono ancora terminati.
- **allow_reuse_address**: Consente di sovrascrivere un indirizzo già utilizzato in precedenza.

Terminazione del server

```
def exit_signal_handler(signal, frame):
    print( 'Exiting http server...')
    try:
      if( server ):
        server.server_close()
    finally:
      sys.exit(0)
exit_signal = signal.SIGINT
signal.signal(exit_signal, exit_signal_handler)
```

Viene definita la funzione **exit_signal_handler** che si occupa di chiudere il server, tale funzione viene chiamata dal segnale **exit_signal** che corrisponde alla pressione della combinazione di tasti **CTRL**+**C**.

Avvio del server

```
try:
  while True:
    print(f"Server http://{HOST}:{PORT} ready...")
    server.serve_forever()
except KeyboardInterrupt:
  pass

server.server_close
```

Il server viene avviato e stampa su terminale il link per poter visionare la pagina web, **serve_forever** permette di gestire un numero illimitato di richieste fino a che non viene segnalato l'evento **KeyboardInterrupt** il quale terminerà il ciclo e chiuderà il server.