ELABORATO RETI

Malagoli Anna A.A. 2023/2024
Matricola: 0001070926

Traccia 2: Web Server Semplice

Creazione di 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.

SISTEMA SERVER CLIENT

Il sistema client-server è un modello di rete molto diffuso per la distribuzione di file di diverso tipo sulla rete da un calcolatore a un altro. Differentemente per quanto avviene nelle reti peer-to-peer, in cui ogni oggetto di rete ha il ruolo sia di server che client, nel sistema client–server ogni nodo nella rete può assumere solo uno tra i due ruoli. In tale sistema è il client che cerca di connettersi al server (indirizzo IP e numero di porta su cui è attivo) per fare richiesta di scaricare uno specifico file. Perché tale operazione possa avvenire è necessario che il server sia in ascolto su tale porta, altrimenti il tentativo di connessione fallirà. È possibile avere diverse tipologie di implementazioni del server legate al numero di client che riesce a gestire contemporaneamente. Il caso più semplice è che il server riesca a soddisfare una richiesta alla volta, ovvero che la connessione con il client che ne ha fatto richiesta venga mantenuta fino alla sua terminazione sulla porta principale in cui il server si mette in ascolto passivo. Se si vuole implementare, come in questo caso, un server che gestisca più richieste contemporaneamente è necessario creare un nuovo thread effimero per ogni client. Sul thread specifico creato dal server per ogni client viene copiata l'intera transazione di handshake che il client ha effettuato sulla porta principale del server. Con l'uso dei thread paralleli il server è sempre accessibile da nuovi client, in quanto la sua porta principale viene lasciata libera di effettuare nuove connessioni creando un thread su richiesta dei client.

ANALISI DEL CODICE SORGENTE

Librerie utilizzate all'interno del programma

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

Sono importati i seguenti moduli:

- La libreria **sys** utilizzata per gestire i dati inseriti nella riga di comando. Permette di utilizzare i metodi e le funzioni della lettura da command line;
- La libreria **signal** che consente di definire handlers personalizzati per effettuare delle operazioni sul programma in esecuzione mediante segnali presi da tastiera;
- La libreria **http.server** alla base della struttura multi-thread che contiene funzioni per la creazione e la gestione di un server web all'interno del programma;
- La libreria **socketserver** che consente di realizzare il socket utilizzato per mettere in ascolto il web server creato. Attraverso tale libreria è possibile effettuare l'operazione di binding della porta e dell'indirizzo IP per fare in modo che siano un unico oggetto su cui il server viene chiamato dai client.

Scelta della porta su cui mettere in ascolto il server web

```
 9  if sys.argv[1:]:
10    port = int(sys.argv[1])
11  else:
12    port = 8080
```

All'interno del programma è definito un costrutto if-else che verifica se sono stati inseriti valori da tastiera. Infatti, è possibile inserire da command line il numero di porta su cui si vuole mettere in ascolto il server multi-thread creato nel programma. Nel caso in cui non venga inserito alcun input da riga di comando viene impostata di default la porta 8080.

Gestione di richieste per servire file statici (come HTML, CSS, immagini)

```
15  server = socketserver.ThreadingTCPServer(('localhost',port), http.server.SimpleHTTPRequestHandler )
```

Viene creato un server TCP multi-thread, con l'host e la porta specificati, che utilizza la classe "SimpleHTTPRequestHandler" per la gestione delle richieste effettuate dai client. In tale modo quando il client effettua la richiesta di accedere ad una risorsa presente sul server viene analizzata la richiesta dalla classe base "BaseHTTPRequestHandler" che implementa le funzioni do_GET() e do_HEAD().

La funzione do_GET() è utilizzata per prelevare il dato richiesto dal client e presente sul server, mentre la funzione do_HEAD() si occupa della gestione delle intestazioni, ovvero legge il content-type presente nella pagina richiesta. In questo modo è possibile inserire all'interno della pagina contenuti diversi dal semplice testo quali ad esempio formati di testo specifici o immagini. Infatti, la pagina HTML oltre a contenere i riferimenti ai link delle risorse presenti nella stessa o in altre directory specifica per ognuna di tali risorse il riferimento alla tipologia del loro contenuto. Senza l'utilizzo di questo metodo non sarebbe possibile fare richieste da parte del client di immagini o file CSS perché non verrebbe considerato il riferimento alla tipologia di contenuto della pagina web.

Il sever viene attivato sulla porta configurata nel dispositivo, ovvero nel local host, che corrisponde all'indirizzo sintattico della macchina stessa. Il local host corrisponde all'indirizzo 127.0.0.1 chiamato indirizzo di loopback e che è l'indirizzo locale che tutti gli host hanno associati alla propria NIC interna, che si occupa della gestione delle comunicazioni interne della macchina.

Il server web realizzato permette inoltre la gestione in simultanea di più richieste da parte dei client mediante l'utilizzo della funzione 'ThreadingTCPServer' contenuta all'interno della libreria socketserver, che permette di creare il socket utilizzato per mettere in ascolto il web server. La funzione 'ThreadingTCPServer' crea sulla porta principale, il cui numero è inserito da tastiera e che di default è la 8080, il socket server con cui viene effettuata l'operazione di handshaking (ovvero la porta su cui il client contatta il server aprendo la connessione) quando un client vuole avviare una transazione con il server. Una volta effettuata tale operazione che consente al socket server di uscire dallo stato bloccante di accept, viene effettuata dal server un'operazione di forking e creato un processo figlio dotato di un Proccess Identifier (PID) autonomo su cui viene mappata la transazione con il client. In questo modo il server è in grado di gestire più richieste da client diversi lasciando sempre libera la porta su cui si mette in ascolto e aprendo, ogni qual volta venga effettuata una

richiesta di connessione, una porta effimera temporanea su cui vengono scambiati i dati con il client.

Gestione dei processi appesi, ovvero non ancora terminati, nel momento in cui il server viene chiuso da tastiera premendo i tasti Ctrl-C

```
22  server.daemon_threads = True
```

Viene impostato il “daemond_threads” a true in modo tale da gestire il comportamento del server nel momento in cui viene improvvisamente arrestato. Tale operazione viene effettuata in modo tale che se il server venisse arrestato improvvisamente, premendo i tasti Ctrl-C, non rimarrebbero dei processi appesi. Se non venisse scritta questa istruzione all’interno del codice sarebbe necessario aspettare del tempo prima di poter mettere nuovamente in ascolto il server sulla stessa porta dopo che si fosse verificato un suo arresto improvviso. Infatti, il sistema operativo vedrebbe ancora impegnata la porta su cui era precedentemente in ascolto il server. Con tale istruzione, invece, il sistema operativo libera immediatamente la porta che risulta essere nuovamente disponibile.

```
25  server.allow_reuse_address = True
```

Impostando “allow_reuse_address” a true il server consente la riutilizzazione del socket anche se non è ancora stato rilasciato quello precedente, andandolo a sovrascrivere.

Ricezione di un segnale

```
29  def signal_handler(signal, frame):
30      print( 'Exiting http server (Ctrl+C pressed)')
31      try:
32        if( server ):
33          server.server_close()
34      finally:
35        sys.exit(0)
```

Definizione della funzione “signal_handler” che consente la gestione del segnale “signal”. Una volta che il programma in esecuzione riceve il segnale “signal” viene visualizzata la stringa che esplicita il fatto che si sta effettuando l’operazione di chiusura del server. Attraverso un try/finally viene chiuso il server se esiste, ovvero se server è a true, e terminato il processo dell’applicazione.

```
38  signal.signal(signal.SIGINT, signal_handler)
```

La funzione “signal.signal” consente di definire handler personalizzati da eseguire quando si riceve lo specifico segnale da tastiera Ctrl-C.

Esecuzione e interruzione del server web

```
42  try:
43    while True:
44      print("Server is starting on http://{}:{}".format('127.0.0.1', port))
45      server.serve_forever()
46  except KeyboardInterrupt:
47    pass
48
49  server.server_close()
```

Una volta avviato il server viene mantenuto in esecuzione all'infinito, mediante un ciclo while true, fino a che non arriva un'interruzione da tastiera, ovvero fino a che non viene segnalato l'evento KeyboardInterrupt. Il metodo "server_forever" effettua un poll interval verificando ogni 0.5 secondi se da tastiera è arrivata una combinazione di tasti di tipo Ctrl-C.

Mediante il metodo "server_close" viene chiuso il server.

ESECUZIONE DEL PROGRAMMA

Per eseguire il file è necessario eseguire il programma in Python ed inserire all'interno di un browser web locale l'URL per visualizzare la pagina web, contenuta nella stessa directory del programma eseguito, che viene scaricata dal server contente un'immagine e delle stringhe di testo formattate.

URL: http://127.0.0.1:8080/pagina.html