Corso di "Programmazione di reti"

Elaborato per il corso: Traccia 2 - Web server semplice

Capponi Federico

Matricola: 0001059826

Indice

1 Obiettivo del progetto

Realizzazione di un semplice Webserver su rete locale:

- **Gestione di richieste GET per file statici**. Il server, una volta avviato da terminale, riesce a presentare al client (o patreon) sia tramite browser che tramite script python dedicato i file presenti nella cartella radice dove viene eseguito. I file supportati sono immagini di vario formato, file HTML e CSS.
- **Gestione richieste simultanee tramite multithreading**. Il server è capace di gestire più richieste contemporaneamente tramite l'uso di programmazione multithreading. Un thread viene aperto per ogni nuova richiesta effettuata al server, che le porta a compimento in modo parallelo, senza intralciare la modalità di utilizzo.
- **Gestione header HTTP e codici di stato**. Il serever riesce a instradare correttamente le richieste valide, quelle errate (come una chiamata ad un file non esistente) e anche eventuali errori di instradamento della connessione. I tre maggiori codici di stato (200, 400, 500) vengono supportati nella comunicazione con i patreons.

2 Soluzione proposta

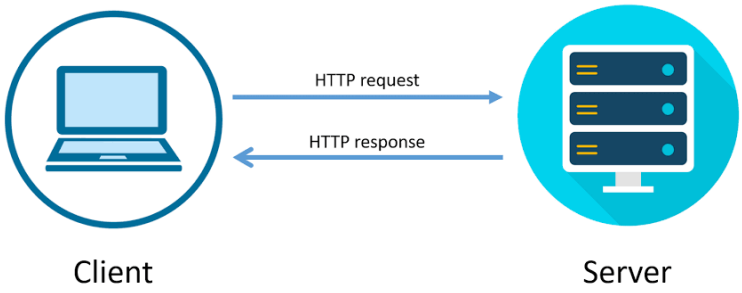

Figura 1: Richiesta HTTP

Librerie Socket e Threading. Nell'implementazione del server e del patreon vengono usate queste due librerie: la prima per creare e gestire il comportamento di un nodo in ascolto, che lavora su un preciso indirizzo IP (tipicamente localhost o 127.0.0.1) e con una porta su cui fa sniffing delle richieste; la seconda, per permettere di suddividere il lavoro generato dalle richieste delegandolo a thread separati, uno per ciascuna richiesta.

Socket_Server.py e Socket_Patreon.py. Il pacchetto zip si compone, oltre a questa presentazione, di due cartelle con all'interno i suddetti file. Con il socket del server sono anche inclusi alcuni file di prova per coadiuvare l'implementazione vera e propria del server. Il file per il client, invece, viene presentato singolarmente in quanto vuole solo essere un'alternativa alla visualizzazione propriamente più corretta: via browser.

Necessità di utilizzo. I due file specificati sopra possono essere usati in modo congiunto, facendo prima avviare il server. Ricordarsi di specificare correttamente le stesse informazioni per indirizzo e porta, in modo da riuscire a stabilire una connessione. Sono comunque impostati valori similari di default per permettere di testare, con facilità, delle connessioni tra dispositivi anche diversi sulla stessa sottorete. Se dovessimo mettere su internet il webserver servirebbe un'implementazione più adeguata, così come un allaccio sul mondo, quindi si vuole specificare in questa sede che questo progetto ha scopo locale.

3 Dettagli implementativi

- **Avvio Server**. Per avviare il server aprire un terminale nella cartella dove si trova il Socket Server ed eseguirlo con "python Socket_Server.py" su Linux o similmente su Windows. Una volta lanciato il comando si possono, opzionalmente, inserire due parametri in input: il primo per l'indirizzo IP, nel quale si potrebbe anche inserire l'IP locale assegnato al terminale dal dispositivo di routing, e il secondo relativo alla porta di ascolto, teoricamente compresa nell'intervallo prestabilito per le porte di connessione (all'atto pratico non è stato inserito nessun controllo nel codice sicché si confida nel buon senso dell'utilizzatore finale). Una volta fatto ciò, il server conferma l'ascolto a video.

- **Avvio Patreon**. La connessione può essere allacciata in due modi: il primo, più tipico di un web server, tramite browser e il secondo, tramite applicativo python incluso nella cartella "patreon". Analizziamo il primo: per connettersi al server basta digitare nella barra degli indirizzi qualcosa del tipo "`http://indirizzoIP:porta/nomefile`". Se l'indirizzo IP fosse stato specificato, si potrebbe permettere anche ad un altro dispositivo nella rete locale dello stesso router di accedere ad uno dei file presenti nella cartella "server" allegata. Per quanto riguarda il secondo metodo, invece si tratta di eseguire sempre con python il file "Socket_Patreon.py" e specificare sempre le tre informazioni di base richieste anche dal metodo precedente.

- **Differenziazione Richieste**. Dentro entrambi i file python è differenziata la ricerca dei file tra immagini e file ipertestuali, filtrando il nome del file richiesto per estensione. Nel caso il file sia di uno dei tipi di immagine più diffusi, il contenuto per essere mostrato correttamente in una pagina web non viene codificato, mentre in tutti gli altri casi si. Questa interpolazione si verifica anche a livello patreon, per permettere di gestire le diverse richieste in modo più opportuno.

4 Problemi incontrati

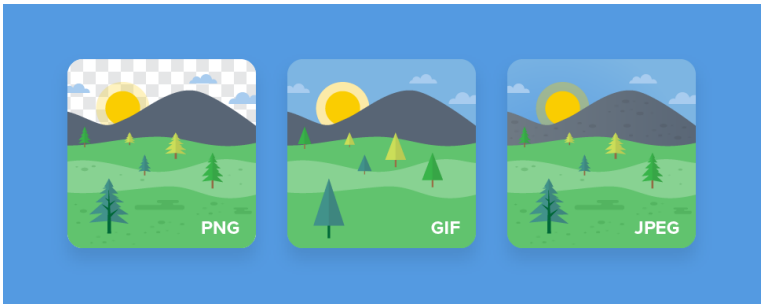

Figura 2: Estensioni immagine supportate

Sono state incontrate principalmente due fattori problematici nello sviluppo degli applicativi:

- **Gestione Immagini**. La prima challenge era quella già precedentemente citata: dover gestire tipi diversi di file. Sia per mostrare su browser che su terminale un output descrittivo e leggibile, si è deciso di non filtrare le immagini con una codifica e di aprire i file conseguentemente in binario. Per la gestione di altri file tipici si è ricorso alla semplice apertura in lettura.

- **Parallelizzazione**. Riuscire a gestire il multithreading è stato il secondo punto chiave: l'implementazione seguita differisce da quella vista a lezione per l'uso della libreria *Threading* con la quale si apre un "filo" per ogni operazione richiesta. Questi sono salvati in un vettore nel quale alla fine della funzione di inizializzazione del server viene effettuata un operazione di join. Si può verificare che l'implementazione funziona anche semplicemente caricando il file *index.html*, che è stato opportunamente modificato per richiedere l'apertura di 3 file contemporanemente (l'immagine, il css e l'html stesso).

5 Conclusioni

Per non appesantire il codice, si può notare che da terminale le risposte del server non sono codificate e vengono stampate in binario, sia le immagini sia file testuali. A livello patreon da terminale invece le risposte sono leggibili come da un visualizzatore di testo. La cosa più interessante del progetto è stata vedere che specificando il giusto IP, ovvero quello della macchina su cui viene fatto partire il server, si può accedere anche da altri terminali facendone così quasi un file sharing site.
Ringrazio chi mi ha supportato con questo progetto: la compagna, un collega di lavoro e i compagni di università.