Web Server Con Adattamento Di Contenuti Statici

Lorenzo Pantano

Ingegneria Di Internet & Web 2019/2020

Lorenzo Pantano
0240471
Università degli Studi di Roma Tor Vergata

Introduzione

Lo scopo del progetto è quello di implementare un Web Server concorrente in grado di gestire dinamicamente l'adattamento di immagini in base alle caratteristiche del dispositivo del client. Per adempiere a questo scopo sono stati usate librerie e API di terze parti come "51 Degrees" per determinare le caratteristiche del dispositivo del client e la libreria "Turbo JPEG" per effettuare operazioni di compressione e ridimensionamento delle immagini.

Struttura del progetto

Il progetto è organizzato nelle seguenti cartelle e file:

- bin
- files
- lib
- src
- Makefile
- README.md
- install.sh

La cartella "bin" conterrà l'eseguibile "main" una volta compilato dal Makefile.
La cartella "files" contiene tutti i file con cui il server risponderà alle richieste dei client (index.html, favicon.ico, immagini).
La cartella "lib" contiene le librerie (statiche e dinamiche) utilizzate per compilare il codice C quali

- fiftyone-common-c.a
- fiftyone-device-detection-c.a
- fiftyone-device-detection-complete.so
- fiftyone-hash-c.a

Inerenti alla device detection di 51 Degrees, e la libreria

- turbojpeg.a

Utilizzata per la compressione di immagini.

La cartella "src" contiene tutti i file .c e .h del progetto. In generale sono presenti i seguenti file:

- main.c/main.h

Punto di inizio del programma, contiene le funzioni per fare il parsing degli argomenti passati a riga di comando e per inizializzare il server.

- **httpHandler.c/httpHandler.h**

Gestisce le connessioni e le richieste da parte dei client

- **img.c/img.h**

Gestisce l'elaborazione delle immagini

- **cache.c/cache.h**

Verifica se un file è presente in cache oppure no

- **errorHandler.c/errorHandler.h**

Gestisce gli errori possibili durante l'esecuzione del server

- **utils.c/utils.h**

Contiene una serie di funzioni utili alle varie componenti del server

- **httpFieldsStruct.h**

Contiene la struttura C che ospita i campi della richiesta http pervenuta

- **userAgentInfoStruct**

Contiene la struttura C che ospita i risultati della Device Detection di 51Degrees

- **51DegreesDeviceDetection/**
- **deviceDetection.c/deviceDetection.h**

Contiene funzioni per la detection del dispositivo usato dal client.

Nella cartella "51DegressDeviceDetection" sono presenti poi tutti i file utili allo scopo di rilevare il dispositivo, compresi file di dati in formato .hash.
Riguardo a questo argomento, si è scelto di usare come metodo di rilevamento quello basato su Hash, in quanto è offline e molto rapido, basandosi su un set di User-Agent conosciuti di circa 20'000 entry.

Implementazione

All'avvio del programma, la prima funzione eseguita è quella che fa il parsing dei parametri da riga di comando:

```
void parseCommandLine(int argc, char const *argv[]);
```

I parametri che possono essere specificati sono il numero di porta su cui sarà in ascolto il server, e l'abilitazione o meno della cache.

Usage: ./main [-p portNumber] [-c dim]

Entrambi i parametri sono opzionali. Il numero di porta di default è 8080.

La cache di default è attiva. Passando come -c dim, -c 0 la cache viene disattivata. Cambiando il numero di porta, la funzione “controlla” prima che il numero di porta sia disponibile. (E’ un controllo molto teorico, il server non va effettivamente a verificare che la porta sia disponibile).
Esempio con porta 9898 e cache disattivata:

./main -p 9898 -c 0

Dopo aver effettuato il parsing, il server si mette in ascolto sulla porta stabilita ed inizia ad accettare connessioni. Le connessioni accettate vengono delegate ad un thread. E’ stato scelto quindi di usare i thread per gestire le connessioni. La motivazione è che la creazione di un thread è un'operazione molto meno onerosa dal punto di vista del sistema operativo, rispetto alla creazione di un nuovo processo tramite fork(), rendendo così il server molto più efficiente nella gestione delle connessioni.

Creazione di un nuovo thread incaricato di gestire la connessione:

```
pthread_t tid;
struct pthread_param *param = malloc(sizeof(struct pthread_param));
param->socket = conn_socket;
strcpy(param->ip_address, client_ip_addr);                          //Client ip
address

if (pthread_create(&(param->tid), NULL, (void *) handleClientConnection, param))
{
   perror("Errore nella creazione del thread");
   exit(EXIT_FAILURE);
}
```

La gestione della connessione viene quindi delegata al thread “handleClientConnection” che legge la richiesta HTTP del client. Quest’ultima viene analizzata e vengono estratte da essa alcune informazioni come lo User-Agent, il fattore di qualità per le immagini e il metodo della richiesta. User-Agent viene utilizzato per fare device detection da 51 Degrees, il fattore di qualità, insieme ad altri parametri ricavati dal dispositivo, viene utilizzato nell’elaborazione di immagini. A seconda del metodo della richiesta (GET o HEAD) viene delegata la risposta ad una funzione specifica, ossia “handleGETRequest” oppure “handleHEADRequest”.

GET Request

Nel gestire il metodo GET, come prima cosa si controlla se il file richiesto è il file index.html , favicon.ico oppure se la cache è disattivata; in tutti questi casi, viene saltata la gestione delle immagini per cui il file viene inviato direttamente senza controllare la cache o senza essere elaborato.
Se il file richiesto è un immagine (nella cartella files/img/) allora si procede in due modi a seconda se il dispositivo rilevato è un dispositivo mobile o desktop (o sconosciuto).

In entrambi i casi però, prima di passare alla compressione effettiva, viene controllata la cache per verificare se il file richiesto sia già stato elaborato precedentemente. In caso di cache hit (il file è presente nella cache) viene mandato questo file. La verifica è fatta tramite la funzione "verifyCache" che non fa nient'altro che aprire la cartella "/files/cache/" e cercare un file con il nome adeguato[1]. Nel caso di cache miss allora si procede con la compressione.

Se il dispositivo è mobile, allora oltre alla compressione si effettua anche un ridimensionamento delle immagini, basato sulla grandezza dello schermo.
La libreria turbojpeg mette a disposizione il ridimensionamento solo per alcuni formati, come ad esempio ½, ⅛, ¼, ecc. per cui va prima calcolato a quale di questi fattori, il rapporto larghezza/altezza schermo del dispositivo mobile, si avvicina di più (funzione "roundToClosestScalingFactor" in utils.c). Vengono quindi instanziate tutte le risorse necessarie per la libreria e viene scritta su disco l'immagine elaborata. A questo punto si può procedere inviando l'immagine compressa. In caso di errori durante la compressione viene inviata comunque l'immagine di qualità base (così come è salvata nella cartella img/).

Nel caso invece di dispositivo desktop o sconosciuto, vengono effettuati gli stessi procedimenti per mobile , solo che questa volta l'immagine viene solo compressa e non ridimensionata.

[1] Con "adeguato" si intende il nome del file calcolato per la cache. Un file nella cache segue la nomenclatura: nomeoriginale-qualità-screenWidth-screenHeight.jpg dove qualità è espresso come numero intero e screenWidth e screenHeight sono posti = 0 in caso di desktop o dispositivo sconosciuto.

HEAD Request

Il metodo HEAD viene gestito allo stesso modo del metodo GET, solo che in questo caso, non viene inviato alcun file. Nello specifico, non viene fatto nessun controllo sulla cache, il file viene aperto ma solo per recuperare le informazioni da inserire nell'header di risposta, come il campo Media Type e il campo Content-Length.

Nota sull' adattamento delle immagini

Le immagini (alcune) che vengono usate in questo server sono di altà qualità già compresse; nelle funzioni di compressione sia per mobile che per desktop queste vengono in realtà prima decompresse e portate alla qualità originale, per poi essere effettivamente ridimensionate. Per cui risulta che alcune versioni delle immagini nella cache siano effettivamente più pesanti di quelle originali. E' stato scelto di lasciare la decompressione dato che per lo scope di questo progetto, nel valutare le prestazioni, la cache viene disabilitata; in una situazione diversa basta evitare di decomprimere le immagini per avere risultati più leggeri in termini di carico da inviare in rete.

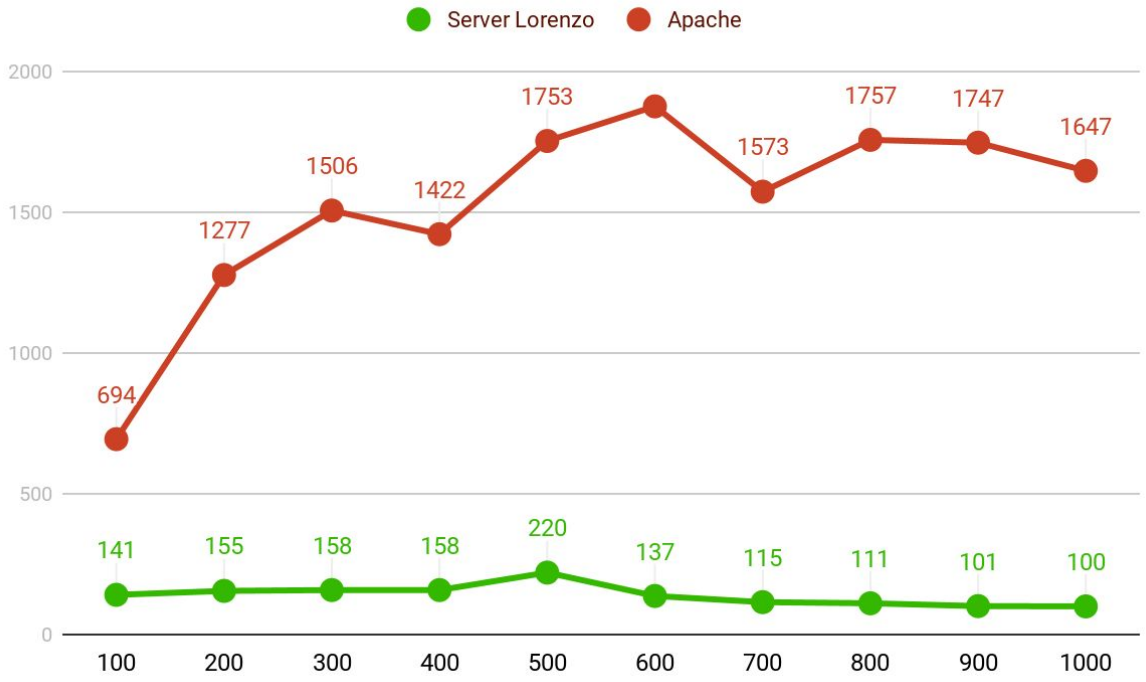
Ambiente di sviluppo e testing

Il server è stato realizzato su una macchina con sistema operativo Windows, ma sfruttando la tecnologia WSL (Windows Subsystem Linux) che permette di avere un sottosistema Linux all'interno dello stesso sistema Windows.
Il codice è stato scritto quindi usando la possibilità messa a disposizione da Visual Studio Code di usare il WSL come ambiente di sviluppo, avendo di fatto i tool del terminale e i tool di build di una macchina Linux.

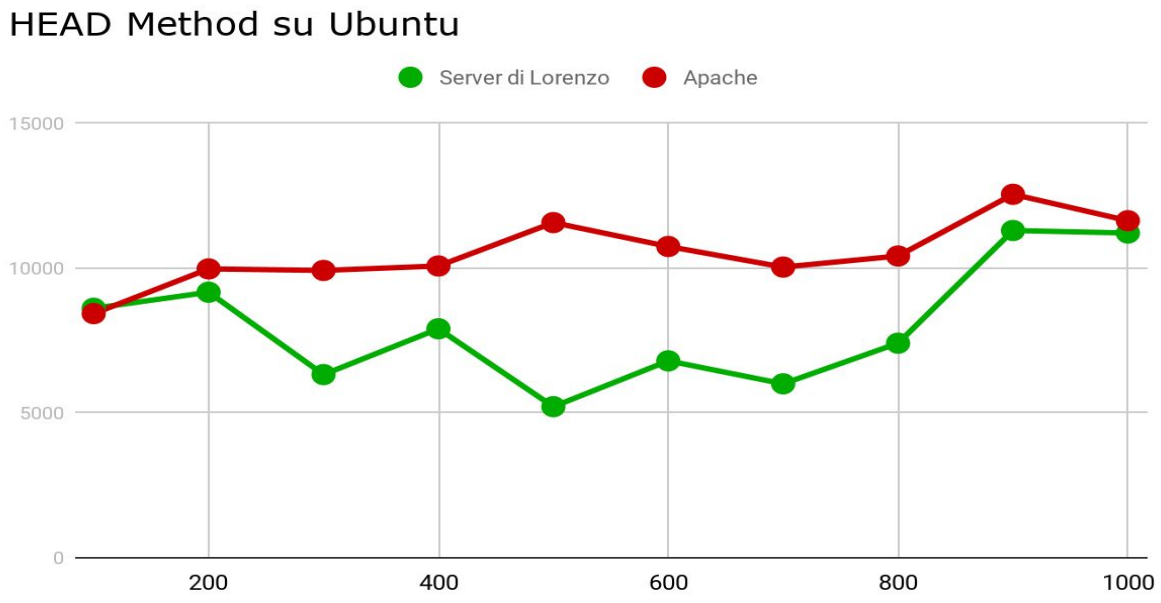
Specifiche della macchina
RAM: 16GB
Processore: Intel Core i7

I test sono stati effettuati a mano usando diversi browser e diverse macchine fisiche, ma anche usando tool come httperf per misurare le prestazioni. Per quanto riguarda queste ultime riportiamo un grafico che le mette a confronto con quelle del web server Apache rispetto ad un numero crescente di richieste di connessioni, testandoli su diverse macchine e sia con metodo HEAD.

Confronto tra “Web Server di Lorenzo” e “Apache” su Windows (GET):



Sull’asse orizzontale troviamo il numero di connessioni effettuate dal tool httperf, sull’asse verticale il connection rate di ciascun server.



Manuale d'uso

Per installare il Web Server su una macchina con sistema operativo Unix seguire i seguenti passi, che verranno comunque inseriti in un file "install.sh" che li eseguirà automaticamente uno alla volta:

- sudo add-apt-repository universe
- sudo apt-get update
- sudo apt install build-essential
- sudo apt install cmake
- sudo apt-get install libturbojpeg
- sudo apt-get install libturbojpeg0-dev
- Estrarre il file device-detection-data.zip nella cartella 51DegreesDeviceDetection/scr in una cartella con lo stesso nome "device-detection-data"
- Tornare nella directory principale del progetto a livello del Makefile ed eseguire make
- cd bin
- Lanciare il programma con ./main [-p portNumber] [-c dim]

Per eseguire tutti i comandi e lanciare il programma direttamente (con parametri di default) eseguire direttamente il file "install.sh"

Se non si riesce ad installare la libreria turbojpeg tramite apt-get la si può installare direttamente:

- git clone https://github.com/libjpeg-turbo/libjpeg-turbo.git
- cd libjpeg-turbo
- cmake -G"Unix Makefiles" -DCMAKE_INSTALL_PREFIX=usr/local
- sudo make install

Esempi di funzionamento

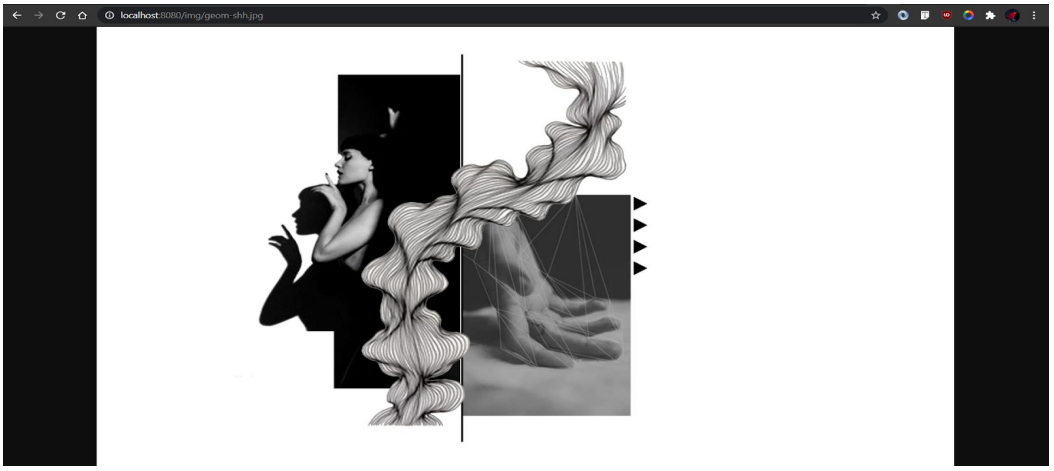
Avviando il server, da un browser si può accedervi digitando l'indirizzo

http://localhost:[porta Selezionata]

Digitando solo quest'indirizzo, si viene reindirizzati sulla pagina index.html, tramite la quale è possibile accedere alle immagini, cliccando sul testo nelle caselle verdi. Il file index.html inoltre si adatta alla visualizzazione mobile, disponendo i contenuti in colonna.

index.html
Browser Chrome (Desktop)

geom-shh.jpg
Browser Chrome (Desktop)



index.html
Browser Chrome (Mobile)

Output nel terminale relativo all'immagine geom-shh.jpg in cache (Desktop)

```
Handling GET Request on socket 4, file: ../files/img/geom-shh.jpg
User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36
Detecting...

DETECTION RESULTS :
isMobile: 0
Screen Width: 0
Screen Height: 0
VERIFY CACHE: geom-shh-80-0-0.jpg
CACHE HIT (NOT MOBILE)
BUILD CACHE NAME
BUILD CACHE NAME IMGNAME: geom-shh.jpg
BUILD CACHE NAME CACHENAME: geom-shh-80-0-0.jpg
FILE TO BE SENT ALREADY IN CACHE (NOT MOBILE): ../files/cache/geom-shh-80-0-0.jpg
File size: 140593
Media type: image/jpeg
Sent succesfully to client: 127.0.0.1 on socket: 4
```

Output nel terminale relativo all'immagine ragazza-museo1.jpg in cache (Mobile)

```
Handling GET Request on socket 4, file: ../files/img/ragazza-museo1.jpg
User Agent: Mozilla/5.0 (Linux; Android 9; Redmi Note 7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.127 Mobile Safari/537.36
Detecting...

DETECTION RESULTS :
isMobile: 1
Screen Width: 1080
Screen Height: 2340
VERIFY CACHE: ragazza-museo1-80-1080-2340.jpg
CACHE HIT (MOBILE)
BUILD CACHE NAME
BUILD CACHE NAME IMGNAME: ragazza-museo1.jpg
BUILD CACHE NAME CACHENAME: ragazza-museo1-80-1080-2340.jpg
FILE TO BE SENT ALREADY IN CACHE: ../files/cache/ragazza-museo1-80-1080-2340.jpg
File size: 824021
Media type: image/jpeg
Sent succesfully to client: 192.168.50.104 on socket: 4
```

Output nel terminale relativo all'immagine ragazza-museo1.jpg non in cache ed elaborata (Mobile)

```
Handling GET Request on socket 4, file: ../files/img/ragazza-museo1.jpg
User Agent: Mozilla/5.0 (Linux; Android 9; Redmi Note 7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.127 Mobile Safari/537.36
Detecting...

DETECTION RESULTS :
isMobile: 1
Screen Width: 1080
Screen Height: 2340
VERIFY CACHE: ragazza-museo1-80-1080-2340.jpg
ACTUAL NAME BEFORE COMPRESSING: ../files/img/ragazza-museo1.jpg
ACTUAL NAME AFTER COMPRESSING: ragazza-museo1.jpg
Is Mobile, Compressing and resizing...
Returning... 1 2
Scaling Factors: 1 2
Compressing okay
ACTUAL NAME COMPRESSION: ragazza-museo1
Writing to disk okay
CACHE MISS (MOBILE)
FILE TO BE SENT: ../files/cache/ragazza-museo1-80-1080-2340.jpg
File size: 824021
Media type: image/jpeg
Sent succesfully to client: 192.168.50.104 on socket: 4
```

Esempio HEAD request

```
Handling HEAD Request on socket 30, file: ../files/index.html
File size: 2481
Media type: text/html
Sent succesfully to client: 127.0.0.1 on socket 30
```