Facoltà di Ingegneria Informatica

Reti di Calcolatori ed Ingegneria del Web

A.A. 2015–2016

Progetto A: Web Server con adattamento dinamico di contenuti statici

G. Cassarà – G. Iannone – E. Savo

Indice

1. Introduzione

Il Progetto “Hideo” è un web server con supporto minimale del protocollo HTTP/1.1, è realizzato in linguaggio C usando le API della socket Berkeley.

Lo scopo è offrire agli utenti la possibilità di scaricare immagini che sono presenti nella pagina principale del server, adattate durante la richiesta in base alle caratteristiche del dispositivo con cui l’utente effettua la richiesta.

Quando un client prova a connettersi al server, esso viene reindirizzato sulla pagina principale, index.html, in cui sono presenti le miniature delle immagini disponibili per il download.

Quando un utente clicca su un’immagine, essa viene convertita automaticamente dal server e poi visualizzata dall’utente nella risoluzione ottimale per il suo device. L’immagine convertita, viene poi salvata dal server in una cache, da cui potrà essere servita immediatamente nelle successive richieste, risparmiando il carico e diminuendo i tempi di risposta.

Il web server ha un’architettura multi-thread con pool di thread statico, quindi è capace di gestire più connessioni simultaneamente. Inoltre, adotta un sistema di logging basato su livelli per discriminare i vari tipi di messaggi: information, warning, error.

L’idea alla base delle scelte progettuali sono state effettuate cercando di ridurre al minimo i tempi di risposta del server.

Ulteriori dettagli verranno analizzati nei successivi paragrafi.

Il codice sorgente è disponibile al seguente indirizzo:

https://github.com/v2-dev/hideo

2. Comportamento dell'applicazione

2.1 Avvio del server

All'avvio del programma vengono letti i parametri porta d'ascolto, numero di thread, backlog e livello di log dal file di configurazione server.cfg e viene creata la socket d'ascolto. Avviene l'inizializzazione di wurfl e della struttura dati della cache e del logger.

Vengono quindi creati i worker thread che si occuperanno di servire il client.

Il server entra in un loop in cui accetta continuamente le connessioni dalla socket d'ascolto e inserisce il file descriptor della connessione in cima ad una lista che verrà letta dai worker thread a seguito della segnalazione del server.

2.2 Gestione client

I worker thread ricevono la segnalazione dal server che un nuovo file descriptor è stato aggiunto nella lista. Quindi uno dei thread prende l'esclusività sul file descriptor, lo elimina dalla lista e serve permanentemente il client sulla stessa socket (permanenza della connessione del protocollo HTTP/1.1).

2.3 Processamento della richiesta

Il thread effettua un parsing della richiesta http. Se il parsing dà esito positivo - ovvero il client rispetta le specifiche del protocollo HTTP/1.1 - passa lo User-Agent a WURFL, il quale restituisce le dimensioni dell'immagine ottimizzata.

Inoltre il thread memorizza i seguenti parametri per l'immagine: il fattore di qualità e l'estensione richiesta per l'immagine.

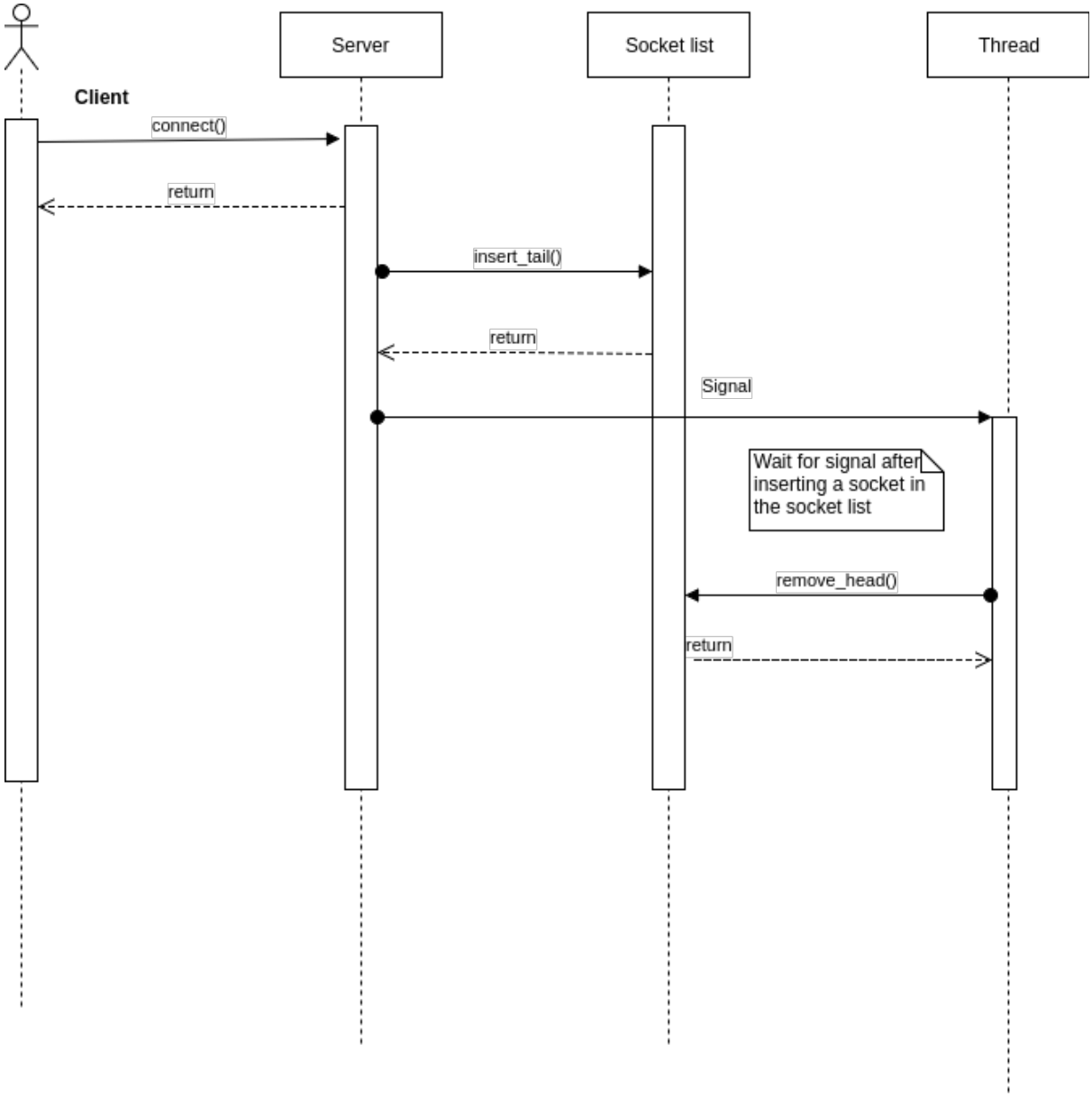
Controlla che il file richiesto dall'utente (con le nuove dimensioni) sia in cache :

(a) Se il file si trova in cache viene inviato direttamente, senza effettuare alcuna operazione di conversione dell'immagine.

(b) Se l'immagine non si trova in cache, essa viene convertita dal programma ImageMagick, inviata al client e salvata successivamente nella cache;

2.4 Chiusura della connessione

La connessione verrà chiusa nel momento in cui il client si disconnette oppure è scattato il timeout della connessione senza aver ricevuto nulla dal client.



3. WURFL

Per quanto riguarda l'adattamento dinamico delle immagini abbiamo adoperato le API esterne di WURFL, un progetto sviluppato da ScentiaMobile. L'API WURFL legge l'intestazione della richiesta HTTP generata dal dispositivo estraendo lo User-Agent generato dall'utente. Successivamente, l'API WURFL utilizza un algoritmo di matching per identificare il dispositivo ed estrarre le sue caratteristiche (es. Modello di dispositivo, browser, OS, larghezza dello schermo, ecc.) da un database chiamato WURFL.xml.

Per gli scopi della nostra applicazione ci è interessato principalemente ottenere la dimensione dello schermo del device richiedente l'immagine. Con questi risultati possiamo dare all'utente un'esperienza ottimale del nostro web server.

Wurfl è un software proprietario, quindi per ottenere una prova gratuita del programma bisogna fare un'apposita richiesta sul sito https://www.scientiamobile.com/ e richiedere la Trial di "WURFL InFuze C API". Dopodichè si verrà contattati da un operatore dell'azienda, per assicurarsi che la copia del programma verrà usata solo a scopi didattici e non commerciali. Al link https://docs.scientiamobile.com/documentation/infuze/infuze-c-api-user-guide è disponibile la documentazione per avere spiegazioni sull'utilizzo delle loro API. A noi è stato fornito un pacchetto .deb, che permetteva l'installazione del programma in distribuzioni GNU/Linux basati su Debian.

L'ambiente WURFL funziona nel seguente modo: viene inizialmente caricato in ram il Database del programma, un file .xml. Il Database è rappresentato in C da una variabile opaca wurfl_handle, che appunto si può interrogare mediante delle API, per richiedere le caratteristiche di un dispositivo con un certo User Agent. Leggendo la documentazione del sito si legge la seguente nota:

"WURFL engine is thread safe and the intended design is that multiple threads share the same wurfl_handle."

Ciò significa che il wurfl_handle è progettato per rispondere a più interrogazioni provenienti da diversi thread, senza alcun problema. Essendo il nostro Web Server multi-thread, questa caratteristica è stata fondamentale per semplificare la gestione della concorrenza.

Cominciando ad utilizzare la libreria Wurfl, abbiamo notato subito un problema: a quasi tutti i nostri User Agent di test veniva data una risoluzione in larghezza di 800 pixel, e in altezza di 600 pixel, che è la risoluzione di default che forniscono le API Wurfl quando non riescono ad identificare il dispositivo. Infatti, aprendo il file .xml, abbiamo notato che, almeno nella versione che ci è stata fornita, erano presenti soprattutto User Agent di dispositivi mobili. In questo modo sarebbe stato impossibile fare (in maniera rapida) i test sull'adattamento dinamico delle immagini del nostro Web Server! Per risolvere questo problema abbiamo scaricato un addon per Firefox al link https://addons.mozilla.org/it/firefox/addon/user-agent-switcher/ che permette al browser di "apparire" con un User Agent differente, e quindi ci ha permesso di impostare degli User Agent che fossero presenti nel database di Wurfl.

4. Adattamento dell'immagine

Per convertire le immagini abbiamo utilizzato il programma esterno ImageMagick, chiamando il comando convert dalla nostra applicazione. Poniamoci nell'ipotesi in cui il client richieda una risorsa non presente nella cache: viene effettuato il parsing del messaggio HTTP di richiesta, al fine di reperire,tramite lo User Agent(ottenuto grazie a Wurfl), i parametri di larghezza-altezza, mentre la qualità è ottenibile dal campo Accept del messaggio, insieme anche all'estensione desiderata dal client. Questi parametri diventano input della funzione nConvert, la quale si occupa di preparare la stringa con il comando convert ed i relativi parametri, da passare poi alla chiamata di sistema system().

Abbiamo scelto di utilizzare system perché esso crea attraverso una fork un processo figlio che si occuperà della conversione dell'immagine, diminuendo il carico di lavoro del thread worker. La nuova immagine viene quindi inviata al client, e una copia di questa viene salvata direttamente nella cache.