RELAZIONE LABORATORIO INCAPACHE

Laboratorio eseguito da:
-Cuneo Giulio: s4516855
-Magno Alessandro: s4478234

3.0

Partendo dalla versione 3.0 di questo laboratorio, ne abbiamo verificato il corretto funzionamento effettuando un confronto passo passo tra la nostra esecuzione e quella dell'eseguibile fornitoci.

Per quanto riguarda il parsing delle stringhe, che riguardano date e orari, l'abbiamo testato facendo stampare i vari contenuti di variabili e strutture, sia per capire come gestirle, sia per constatarne la correttezza.

Una volta concluso il completamento del codice, la parte di debugging è iniziata verificando il funzionamento della memoria cache, nel senso che cercando una pagina già visitata in precedenza, abbiamo fatto in modo che il nostro browser caricasse una pagina che precedentemente aveva già salvato in memoria.
Ci siamo quindi serviti di un file "self.html" che abbiamo lasciato nella cartella "www-root", con l'indirizzo della pagina in questione (in sostanza una pagina che "punta a se stessa"), così da evitare che premendo il tasto "aggiorna" il browser inviasse una nuova richiesta.

Un altro test eseguito è stato il controllo nel caso in cui la pagina già cercata, avesse subito delle modifiche e quindi il browser avrebbe dovuto aggiornare la nostra copia.
Per far ciò, una volta cercata la pagina, abbiamo modificato il suo file .html (contenuto nella directory del server) in modo tale da cambiare la data di ultima modifica: provando a ricaricare la pagina il browser si accorge che esiste una versione più recente ed invia una nuova richiesta, aggiornando la copia presente in cache.

Un altro test da noi condotto, è stato quello di verificare che l' UserID cambiasse ogni volta che il browser si fosse connesso ad una pagina per la prima volta. Ci siamo quindi connessi ad un indirizzo, abbiamo aperto un'altra pagina (viene mantenuto lo stesso UserID), dopodichè abbiamo cancellato i cookie e abbiamo eseguito una riconnessione: l'UserID è stato ora incrementato correttamente.

La difficoltà maggiore che abbiamo riscontrato, è stata l'inserimento permanente dei cookie.
Durante i test sulla permanenza (una volta chiusa la sessione di browser questi sarebbero dovuti rimanere in cache ed essere cancellati solo alla scadenza di una data fissata) abbiamo notato che browser diversi operano in modo diverso, ad esempio firefox di default tiene in cache la pagina visitata per 5-7 minuti ed è usufruibile anche in un' altra sessione di browser con il webserver chiuso, mentre in chrome , una volta chiuso il webserver, la pagina non risulta più disponibile.
Per avere la certezza che i cookie non venissero cancellati prima della scadenza, abbiamo provato il programma con inserita la "Expires date", cercato una pagina, abbiamo poi chiuso browser, webserver e spento il pc. Alla riaccensione, siamo andati direttamente sul browser (nel nostro caso firefox), abbiamo inserito il link e la pagina era ancora disponibile.

3.1

La versione 3.1 è stata ancor più impegnativa perché è molto difficile testare se l'implementazione è effettivamente corretta, in quanto in un programma multi-thread il comportamento potrebbe cambiare ad ogni esecuzione.

Per testare il multi-threading abbiamo optato per utilizzare l'operatore "|" nella bash:
in questo modo abbiamo potuto eseguire più richieste in successione, evitando che l'intervallo di tempo intercorso tra una e l'altra fosse abbastanza ampio da permettere lo svuotarsi (join_prev) della coda dei thread.

In particolare, dal momento che abbiamo a disposizione 16 slot per il multi-threading (8 per l'ascolto, 8 per la risposta, 16 aggiuntivi), abbiamo utilizzato il seguente metodo:
Abbiamo piazzato nella cartella "www-root" del nostro server 8 immagini differenti, ed eseguito un comando del tipo " firefox nome0.jpg | firefox nome1.jpg | …….. | firefox nome7.jpg ".
Tutte le immagini sono state caricate correttamente e, una volta riempiti gli slot necessari e chiamata la join_all i thread sono stati ripristinati in vista di nuove richieste.
Successivamente, abbiamo ripetuto questo esperimento con un numero maggiore di richieste (16 immagini *che alleghiamo* e quelle già presenti all'interno di www-root) , in modo da cercare di saturare tutti i thread a nostra disposizione, così facendo abbiamo avuto la conferma della corretta gestione delle risorse disponibili.
A volte può succedere che una pagina impieghi più tempo a caricare, in quanto, una volta riempiti tutti i thread, deve attendere che si liberi uno slot per poter inviare la risposta.

NB: *comando utilizzato (immagini in "www-root")-->firefox localhost:2000 | firefox localhost:2000/uncadunca.jpg | firefox localhost:2000/1.jpeg | firefox localhost:2000/2.jpeg | firefox localhost:2000/3.jpg | firefox localhost:2000/4.jpeg | firefox localhost:2000/5.jpg | firefox localhost:2000/6.jpeg | firefox localhost:2000/7.jpg | firefox localhost:2000/8.jpg | firefox localhost:2000/9.jpg | firefox localhost:2000/9.jpg | firefox localhost:2000/10.jpg | firefox localhost:2000/11.jpg | firefox localhost:2000/12.jpg | firefox localhost:2000/13.jpg | firefox localhost:2000/14.jpg | firefox localhost:2000/15.jpg | firefox localhost:2000/16.jpg*

NB2: Per utilizzare correttamente il comando si rivela necessario avere almeno una scheda già aperta di firefox.

Consegna precedente

Per quanto concerne l'utilizzo di un metodo differente nelle richieste (come per esempio POST) abbiamo notato che, in modo del tutto casuale, possono verificarsi delle corse critiche per cui il nostro server fallisce il caricamento della schermata di errore, oppure se riesce a caricare la pagina di errore, abbiamo comunque un errore di segmentazione.

Aggiornamento

Nella versione più recente che abbiamo consegnato, penso di aver risolto l'errore di segmentazione e le corse critiche che si presentavano utilizzando il metodo POST. Ho effettuato nuovi test, lascio di seguito la stringa di comando:

firefox localhost:2000 | firefox localhost:2000/uncadunca.jpg | firefox localhost:2000/1.jpeg | firefox localhost:2000/2.jpeg | firefox localhost:2000/3.jpg | firefox localhost:2000/4.jpeg | firefox localhost:2000/5.jpg | firefox localhost:2000/6.jpeg | firefox localhost:2000/7.jpg | firefox localhost:2000/8.jpg | firefox localhost:2000/9.jpg | firefox localhost:2000/10.jpg | firefox localhost:2000/11.jpg | firefox localhost:2000/12.jpg | firefox localhost:2000/13.jpg | firefox localhost:2000/14.jpg | firefox localhost:2000/15.jpg | firefox localhost:2000/16.jpg | firefox localhost:2000/17.jpeg | firefox localhost:2000/18.jpeg | firefox localhost:2000/19.jpg | firefox localhost:2000/20.jpg | firefox localhost:2000/21.jpg | firefox localhost:2000/22.jpg | firefox localhost:2000/23.jpg | firefox localhost:2000/24.jpg | firefox localhost:2000/25.jpg | firefox localhost:2000/26.jpg | firefox localhost:2000/27.jpg | firefox localhost:2000/28.jpg | firefox localhost:2000/29.jpg | firefox localhost:2000/30.jpg | firefox localhost:2000/31.jpg | firefox localhost:2000/32.jpg | firefox localhost:2000/33.jpg | firefox localhost:2000/34.jpg | firefox localhost:2000/35.jpg | firefox localhost:2000/36.jpg | firefox localhost:2000/37.jpg | firefox localhost:2000/38.jpg | firefox localhost:2000/39.jpg | firefox localhost:2000/40.jpg

Considerazioni

Da questa esperienza abbiamo compreso la complessità nel gestire programmi multithread, quindi abbiamo concluso che è meglio evitare di utilizzare i thread se non sono necessari, ma in caso non possiamo evitarne l'utilizzo, occorre investire parecchio tempo nel debugging del codice.