Oggi ci è stato assegnato il laboratorio IncApache 5.0, cominciamo a descriverlo. È diviso in due parti, una obbligatoria (5.0) e una facoltativa (5.1). Queste due versioni corrispondo ad utilizzare i protocolli htpp 1.0 e 1.1.

Vediamo la parte 1.0. C’è un file chiamato main.c, nel quale abbiamo l’avvio del processo. Inizialmente abbiamo l’avvio del processo server web, che attraverso una fork crea un ulteriore processo, destinato all’esecuzione di “file -i”. I due processi sono collegati tra loro da una pipe. Il server verrà creato (compilato) tramite il comando sudo (in modo da poter utilizzare i bit set userId). Per questo motivo quando lanciamo il make è chiesta la password. Se non abbiamo accesso come root alla nostra macchina possiamo compilarlo usando l’opzione “pretend to be root”. Tra le varie opzioni ci sono “debug” e quella per attivare la compilazione delle parti della versione opzionale. Nella cartella www-root ci sono dei file di html per testare.

Nei file c’è da completare i TO BE DONE (che si distinguono in 5.0 e 5.1).

Oltre al main.c, c’è un threads.c che implementa il meccanismo multi-thread. Tra la versione 5.0 e 5.1 cambiano il numero di thread attivati: nella versione 5.0 abbiamo un thread attivato per ogni richiesta da parte di un client. In realtà il numero di thread attivati è un numero fisso, e sono allocati.

C’è poi un file aux.c, in cui da completare ci sono un paio di cose relative al formato standard delle date (serve per realizzare il conditional GET).

La parte più complicata si trova in un file chiamato http.c. Qui ci sono due funzioni, una per fare il parsing delle richieste e un’altra per rispondere al client. Oltre a queste, ci sono funzioni ausiliarie per gestire il meccanismo di cookies, usati dal server per tracciare un client. Ad ogni cookie di identificazione è associato un contatore di tracciamento, che viene incrementato ad ogni richiesta del client.   
I cookies utilizzati devono essere persistenti (quindi devono rimanere validi fino alla loro data di scadenza), noi dobbiamo assegnarvi la data di scadenza.

Il tracciamento avviene in maniera semplice. È stato definito un array di dimensioni predefinite (quindi si possono riconoscere solo n client diversi), se arriviamo alla fine dell’array possiamo sostituire un vecchio client con un nuovo client. Le operazioni di incremento saranno in modulo la dimensione dell’array.

Nel parsing delle richieste abbiamo una prima riga che deve contenere il metodo, il nome del file e la versione (che dovrebbe essere http/1.0 o http/1.1) I metodi usati dal nostro server sono il metodo GET e il metodo HEAD (non riconosciamo il metodo POST, quantomeno nella 1.0). Per quanto riguarda il nome del file basta riconoscerlo e poi cercarlo nel percorso, passandolo al processo file ausiliario per ottenere il MIME type (e passato anche al file STAT per conoscere i suoi metadati). I delimitatori nella riga sono gli spazi vuoti (i blank).

Attenzione al multithreading. Ci saranno funzioni che possono essere usate tranquillamente e altre che causano punti critici (perché non sono rientranti).

Dopo aver fatto il parsing della prima riga, occorre riconoscere le opzioni. Va riconosciuta l’opzione IfModifiedSince (se presente), che deve contenere (dopo i : ) una data nel formato GMT (risposte possibili: 200 OK, 304 Not Modified). Se non presente dovremo sempre rispondere 200 o 404.  
L’altra opzione da controllare è Cookie, che può contenere dopo i :, ad esempio, UserId=Num: se si trova questa opzione va incrementato il contatore della tabella di tracciamento nell’indice corrispondente all’ID, se questa opzione manca, bisogna generare un nuovo UserId e assegnarlo al client.

All’interno di www-root ci sono anche i file per le segnalazioni di errore (che vanno inviati se le richieste contengono degli errori, assieme al messaggio 400 qualcosa).

Assolutamente è da evitare la terminazione del server in caso di errori da parte del client.  
Se un’opzione è scritta in maniera incorretta, allora conviene piuttosto far finta che quell’opzione non ci sia e provare comunque a servire il client.

La risposta (http.c) deve avere un formato ben definito: deve esserci il risultato della richiesta (un numero in formato ascii, che se va tutto bene sarà 200, altrimenti sarà 304 o 400 o 404 ecc.) e sulla stessa riga c’è la spiegazione testuale del significato. Dopodichè abbiamo gli header come opzioni, con il Mime-type, il Last-modified (seguiti dai : ), ecc.   
Sempre attenzione al fatto che il server è multithread, mentre il processo del file -i è single thread, quindi dobbiamo fare in modo che su quello arrivi una sola richiesta per volta, quindi l’accesso alle pipe dovrà essere di mutua esclusione (mutex). Dunque, la preparazione dell’header mime-type avviene in mutua esclusione.  
L’header http è terminato dalla riga vuota, dopodichè abbiamo un body, che contiene un file in caso di richiesta GET (ma non in caso di richiesta HEAD). Per mandare il file possiamo usare la system call chiamata sendfile (che potrebbe dover essere chiamata più volte se non riesce a mandare il file tutto in una volta).

Nel caso della versione 5.1 i problemi saranno maggiori perché la sincronizzazione tra thread sarà più complessa.

Come si fa a mandare in esecuzione più thread contemporaneamente per debuggare il programma? Si devono mandare tante richieste di connessione in contemporanea, per fare ciò basta usare come client un browser e mettere nel server file html che contengano tante immagini. Un altro modo per simulare tante richieste simultanee sarebbe prepararsi uno script che apra una connessione Telnet sulla porta 80.

Se teniamo attiva la cache sul browser, questo farà sempre delle GET condizionali, il problema è che il browser se riceve 304 fa vedere la copia che ha in cache, altrimenti fa vedere il file nuovo, il problema è che in questo modo o cambiamo il contenuto dei file quando li apriamo oppure da browser non notiamo la differenza tra 304 e 200 (altrimenti si può usare una connessione Telnet).

Dovremo consegnare i sorgenti completati e una relazione, su ciò che non è direttamente deducibile dal codice sorgente.

L’RFC prevede che il server possa rispondere al client con una versione minore o uguale alla versione del client (l’importante è che non sia maggiore). Quindi non ci sono problemi se il nostro server risponde al client con una versione di http 1.0 o 1.1.