Laboratorio IncApache

Davide Scarrà

December 31, 2020

1 Specifiche

Specifiche macchina Client:

- **SO:** Ubuntu 20.04.1 LTS
- **CPU:** AMD A10-6800k

2 Introduzione

Questo laboratorio è stato implementato in collaborazione con Lorenzo La Corte, Tommaso Coronati, Milo Galli, Samuele Pignone e Luca Ferrari.
E' stata implementata solo la parte obbligatoria.

Nel terzo e ultimo laboratorio abbiamo implementato un server web, incApache, in grado di gestire e rispondere alle richieste di tipo HEAD e GET.
Il codice è diviso in quattro file:

1. **incApache_main.c:** Riceve in input il percorso della directory da usare come root e il numero di porta sulla quale mettersi in ascolto, crea i socket e gestisce le richieste mediante thread.

2. **incApache_http.c:** Gestisce le richieste http e invia le risposte, si occupa inoltre di controllare la bontà delle richieste, inoltrando opportuni messaggi di errore (es. errore 404 nel caso in cui la pagina non venga trovata e 501 nel caso in cui il metodo richiesto non sia implementato).

3. **incApache_aux.c:** Contiene le funzioni ausiliarie.

4. **incApache_threads.c:** Utilizzato per la gestione multithread (completata solo la parte 4.0).

3 Problematiche

Uno dei primi problemi che abbiamo riscontrato è stato l'esecuzione del file *incapache4.0debug-ubuntu64* che per essere eseguito doveva avere dei permessi particolari. Siamo riusciti ad assegnarglieli con i comandi:

sudo chown root:root incapache4.0debug-ubuntu64
sudo chmod -R 6755 incapache4.0debug-ubuntu64

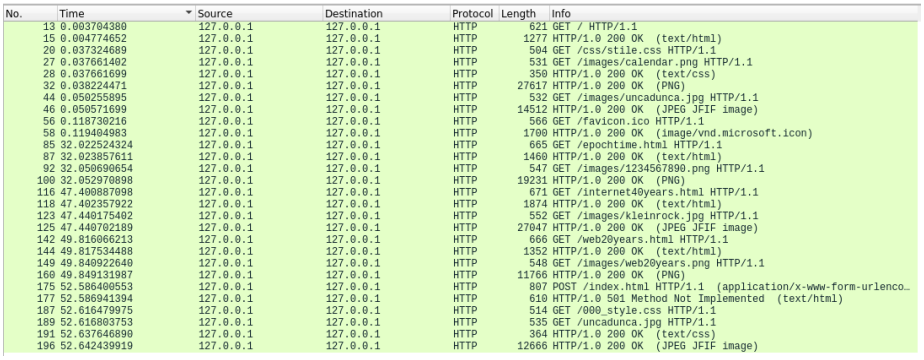
Un'altra difficoltà che abbiamo avuto è stata la *Set-Cookie* perché non riuscivamo ad inserire il valore di *UserID* all'interno della stringa.
Inizialmente quello che abbiamo fatto è stato un ciclo che andava ad inserire dopo al carattere '=' il valore di cookie ma questo funzionava solo per i valori minori di 10 quindi abbiamo risolto spezzando la stringa in corrispondenza dell'uguale, inserendo il valore di cookie e riunendo la stringa.
Un ultimo problema è stato in fase di parsing del cookie per prelevare *option_val* e *option_name* perché il programma andava in segmentation fault, ma abbiamo risolto facendo una *malloc()* prima di memorizzare i valori nelle rispettive variabili. Per convertire il valore del cookie da stringa ad intero abbiamo usato la funzione *atoi()*.
Per il resto non abbiamo riscontrato particolari problemi.

4 Debug

Per il debug abbiamo testato il nostro programma su distro di Linux (Ubuntu e Xubuntu) e browser diversi (Chrome e Firefox) cancellano ogni volta i cookie e la cache. Riguardo i cookie il loro valore aumenta ogni volta che, dopo averli cancellati dal browser, ricarichiamo la pagina web.
Abbiamo utilizzato **Telnet** per inviare singolarmente le richieste al server e verificare la correttezza della sua risposta.

5 Risultati

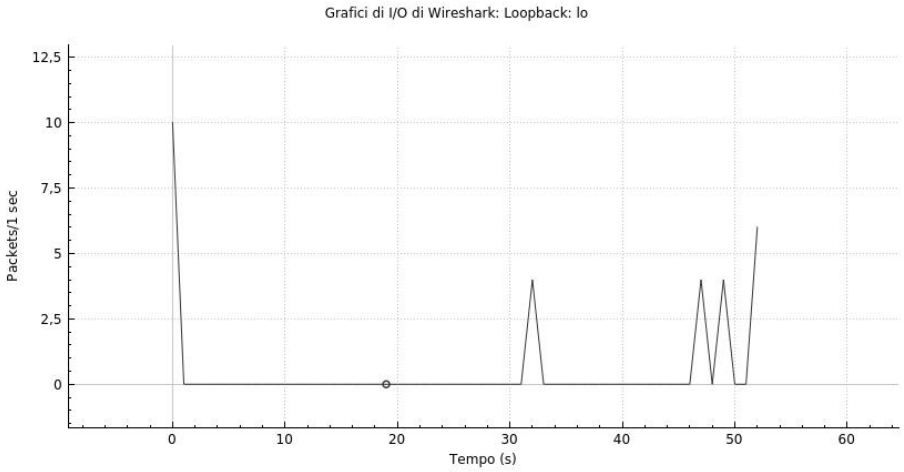
La tabella seguente mostra i pacchetti in entrata e uscita dal client ottenuti analizzando la rete con il programma **Wireshark**.



No.	Time	Source	Destination	Protocol	Length	Info
13	0.003704380	127.0.0.1	127.0.0.1	HTTP	621	GET / HTTP/1.1
15	0.004774652	127.0.0.1	127.0.0.1	HTTP	1277	HTTP/1.0 200 OK (text/html)
20	0.037324689	127.0.0.1	127.0.0.1	HTTP	504	GET /css/stile.css HTTP/1.1
27	0.037661402	127.0.0.1	127.0.0.1	HTTP	531	GET /images/calendar.png HTTP/1.1
28	0.037661699	127.0.0.1	127.0.0.1	HTTP	350	HTTP/1.0 200 OK (text/css)
32	0.038224471	127.0.0.1	127.0.0.1	HTTP	27617	HTTP/1.0 200 OK (PNG)
44	0.050255895	127.0.0.1	127.0.0.1	HTTP	532	GET /images/uncadunca.jpg HTTP/1.1
46	0.050571699	127.0.0.1	127.0.0.1	HTTP	14512	HTTP/1.0 200 OK (JPEG JFIF image)
56	0.118730216	127.0.0.1	127.0.0.1	HTTP	566	GET /favicon.ico HTTP/1.1
58	0.119404983	127.0.0.1	127.0.0.1	HTTP	1700	HTTP/1.0 200 OK (image/vnd.microsoft.icon)
85	32.022524324	127.0.0.1	127.0.0.1	HTTP	665	GET /epochtime.html HTTP/1.1
87	32.023857611	127.0.0.1	127.0.0.1	HTTP	1460	HTTP/1.0 200 OK (text/html)
92	32.050690654	127.0.0.1	127.0.0.1	HTTP	547	GET /images/1234567890.png HTTP/1.1
100	32.052970898	127.0.0.1	127.0.0.1	HTTP	19231	HTTP/1.0 200 OK (PNG)
116	47.400887098	127.0.0.1	127.0.0.1	HTTP	671	GET /internet40years.html HTTP/1.1
118	47.402357922	127.0.0.1	127.0.0.1	HTTP	1874	HTTP/1.0 200 OK (text/html)
123	47.440175402	127.0.0.1	127.0.0.1	HTTP	552	GET /images/kleinrock.jpg HTTP/1.1
125	47.440702189	127.0.0.1	127.0.0.1	HTTP	27047	HTTP/1.0 200 OK (JPEG JFIF image)
142	49.816066213	127.0.0.1	127.0.0.1	HTTP	666	GET /web20years.html HTTP/1.1
144	49.817534488	127.0.0.1	127.0.0.1	HTTP	1352	HTTP/1.0 200 OK (text/html)
149	49.840922640	127.0.0.1	127.0.0.1	HTTP	548	GET /images/web20years.png HTTP/1.1
160	49.849131987	127.0.0.1	127.0.0.1	HTTP	11766	HTTP/1.0 200 OK (PNG)
175	52.586400553	127.0.0.1	127.0.0.1	HTTP	807	POST /index.html HTTP/1.1 (application/x-www-form-urlenco…
177	52.586941394	127.0.0.1	127.0.0.1	HTTP	610	HTTP/1.0 501 Method Not Implemented (text/html)
187	52.616479975	127.0.0.1	127.0.0.1	HTTP	514	GET /000_style.css HTTP/1.1
189	52.616803753	127.0.0.1	127.0.0.1	HTTP	535	GET /uncadunca.jpg HTTP/1.1
191	52.637646890	127.0.0.1	127.0.0.1	HTTP	364	HTTP/1.0 200 OK (text/css)
196	52.642439919	127.0.0.1	127.0.0.1	HTTP	12666	HTTP/1.0 200 OK (JPEG JFIF image)

Come possiamo vedere la prima richiesta è una GET dell'index alla quale il server risponde con 200 OK e con il file *index.html*.
Dopo di che il client richiede come prima cosa il layout della pagina web, contenuto in *style.css*, tutte le immagini che sono presenti nel file *index.html* (*calendar.png* e *uncadunca.jpg*) e infine l'icona del sito.
In seguito abbiamo tutte le richieste di accesso alle pagine del nostro sito web e tutte vanno a buon fine tranne quella in cui provo a inviare il mio indirizzo e-mail. A tale richiesta in fatti il server risponde con *501 Method Not Implemented* allegando l'analogo file HTML contenente la pagina dell'errore.
Questo perchè il nostro web server è in grado di gestire solo le richieste di tipo GET e HEAD quindi quando si vede arrivare una richiesta di tipo POST esso risponde con l'errore 501.
Osservando i tempi in cui vengono poste le richieste e ricevute le risposte si nota che il Round Trip Time è di qualche millesimo di secondo, questo perché il nostro web server è implementato in localhost ovvero è il nostro PC che funge sia da client che server ottenendo così un tempo di comunicazione minimo. Ciò è dimostrato anche dal fatto che l'indirizzo IP sorgente coincide con quello di destinazione (127.0.0.1), il quale è appunto l'indirizzo del localhost.

L'immagine seguente mostra lo scambio dei pacchetti della tabella precedente in relazione con il tempo.



All'inizio della connessione abbiamo un picco di pacchetti, infatti il client sta chiedendo al server la sua icona, l'indice e tutte le immagini presenti al suo interno e il server gli sta rispondendo con i file richiesti.
Dopo di che abbiamo 32 secondi in cui le macchine non si scambiano alcunché fino a quando il client non presenta la richiesta di accesso alla pagina *epochtime.html*, di nuovo nessuno scambio per altri 15 secondi e poi arrivano le richieste di accesso alle pagine *internet40years.htm*, *web20years.html* e infine *501_method_not_implemented.html*.