Oggi vediamo la parte opzionale del laboratorio IncApache. Si tratta di completare le parti che contengono i commenti to be done 5.1. Si ricorda che per compilare bisogna modificare il makefile in modo da compilare con questa versione. Con questa versione la versione dell’http passa dalla 1.0 alla 1.1. A questo punto diventa possibile scegliere tra due versioni diverse per rispondere: l’RFC sancisce di usare la versione più recente tra quelle più disponibili al Client (cioè che non sono superiori alla versione del Client).

Per far funzionare la 1.1 bisogna aggiungere ciò che rende le connessioni persistenti e il meccanismo di pipeline.

Il client aprirà la connessione col syn, il server la accetterà con il syn/ack e il client manderà il primo messaggio con l’ack (seguito da quante altre richieste se ne vuole senza aspettare le risposte). Per far funzionare la pipeline servirà mandare indietro le risposte nello stesso ordine in cui sono arrivate le domande.

All’interno del server abbiamo un sistema multithread. Nella versione 5.0 avevamo 8 thread che permettevano di eseguire le stesse istruzioni e ognuno di essi si occupava di una singola richiesta di un client. Per fare ciò i thread si mettevano in attesa aspettando che arrivasse una connessione e, quando ciò avveniva, si occupavano della risposta per poi tornare liberi e in attesa.

Per realizzare ciò abbiamo usato un sistema di muta esclusione attraverso dei lock per evitare race conditions.

Nella versione 5.1 i thread sono suddivisi in due tipi diversi: quelli che parsano le richieste dei client e altri thread, separati, che preparano le risposte (già nella 5.0 avevamo due momenti separati per il parsing e l’invio ma erano affidati allo stesso thread). Questo ci consente di proseguire ad analizzare tutte le richieste: si analizza la prima e poi, mentre un altro thread prepara la prima risposta, passa ad analizzare la seconda e si va avanti così. Il passo ulteriore è quello di aumentare il numero di thread di risposta, mettendone di più rispetto a quelli che gestiscono le richieste (ci si aspetta che per ogni connessione un client farà più di una richiesta), in modo da preparare tutte le risposte in parallelo.   
Per ottenere ciò è necessario però sincronizzare i thread di risposta, in modo che il primo che parte preparerà la risposta che partirà per prima: serve una coda di tipo FIFO distribuita: quindi non centralizzata, ci si aspetta che ogni thread guardi solo quello che partito prima di lui e si imposti come “completato” solo quando anche quello che sta guardando è completato (ad eccezione del primo thread, ovviamente).

Questo si può realizzare facilmente mettendo nella struttura dati “coda” un campo puntatore che punta al predecessore, lasciando a NULL questo campo nel primo thread. Così facendo, per sincronizzare correttamente i thread basta che, chi ha un puntatore diverso da NULL aspetti la terminazione del thread di cui ha il puntatore “precedente” prima di prendere il controllo del socket per mandare la risposta.

Quanti thread possiamo pensare di predisporre in modo da mantenere l’idea di un numero di massimo di thread eseguibili proporzionato alle capacità dell’hardware (un processore potrà usare contemporaneamente solo 1 thread per ogni core)? La scelta che è stata fatta da Chiola è che ogni thread di richiesta possa sempre far partire almeno un thread di risposta: quindi dovranno sempre restare liberi almeno 1 thread di risposta per ogni thread di richiesta. Dopodichè abbiamo i thread di riserva che sono condivisi e che vengono usati per le connessioni che richiedono tante risposte, una volta che un thread condiviso è preso da un thread di richiesta per formare la propria coda di risposte questo diventa, ovviamente, non disponibile per gli altri.  
Notare che con questo sistema Può capitare che tutti i thread di riserva siano assegnati a una stessa coda, lasciando solo un thread di risposta a ciascuno degli altri thread di richiesta.

In totale i thread di richiesta sono 8, mentre quelli di risposta sono 8 thread privati + 16 condivisi (per un totale di 24 thread di risposta).

Ovviamente le strutture dati (code) dei thread dovranno essere gestite in mutua esclusione per evitare corse critiche.