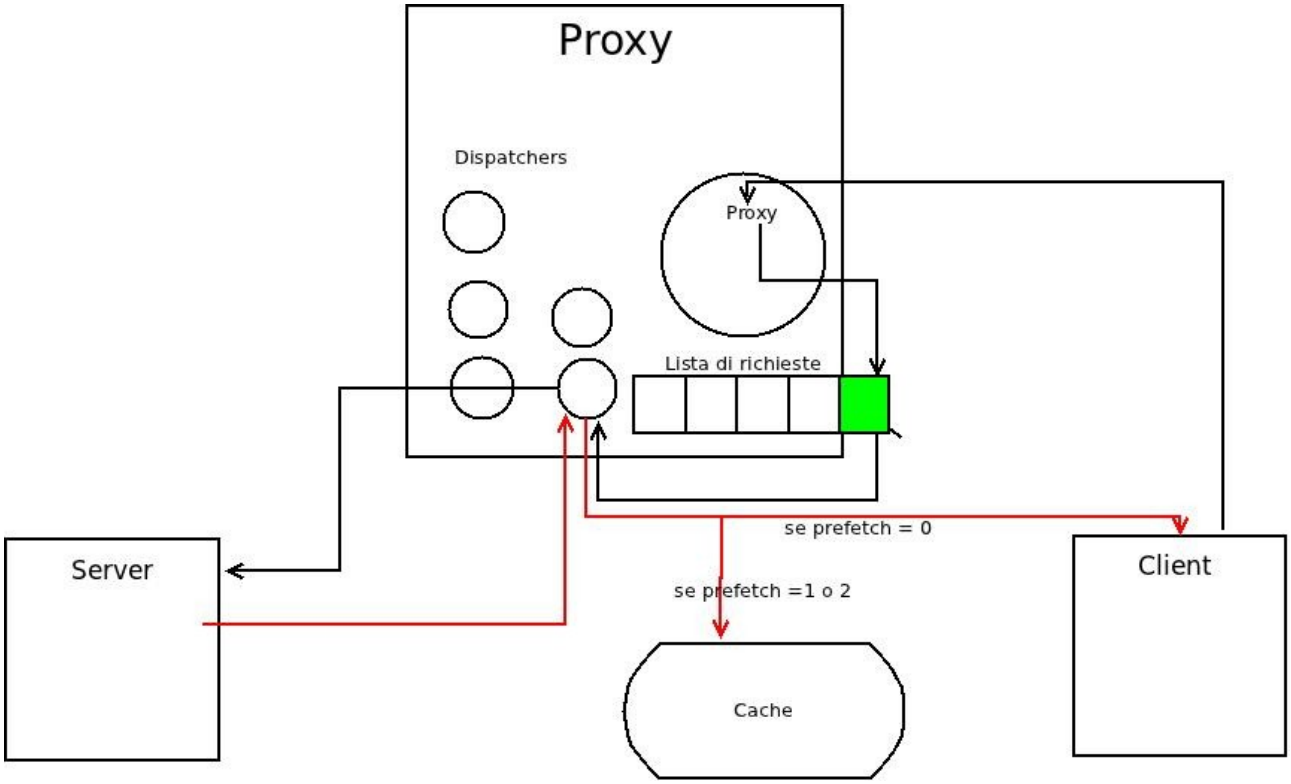
Proxy miniHTTP/miniHTML con pre-fetching

Simone Rondelli, Margherita Lazzarini, Jacopo Giacò

Compilazione

Per compilare il progetto è sufficiente digitare il comando make da terminale.

Struttura generale



Il thread proxy rimane sempre in ascolto di connessioni dal client, e si occupa di mettere le richieste che arrivano dai client nella lista delle richieste. I 5 dispatcher, invece, si occupano di prelevare le richieste dalla lista e le cercano in cache; se non le trovano, le richiedono al server e, una volta ricevute, le rimandano al client oppure le salvano in cache (a seconda del flag prefetch, spiegato più avanti).

File

File che compongono il progetto:

- Proxy.c
- Cache.c, Cache.h
- IOUtil.c, IOUtil.h
- Parser.c, Parser.h
- Request.c, Request.h
- List.h

Proxy.c

Questo file contiene il main del proxy che crea il thread proxy, il quale a sua volta crea i thread dispatcher (5 come da specifiche).
Il proxy prende come argomenti l'indirizzo IP e la porta su cui viene fatta la bind (esempio **./Proxy.exe 127.0.0.1 55554**).
Nel main inoltre vengono richiamate le funzioni *initReq* e *initCache* che inizializzano le strutture dati per la gestione delle richieste della cache (spiegate più avanti).

Il thread proxy dopo aver creato i dispatchers si connette al client (effettuamdo la socket, la bind, la listen e la accept), rimanendo sempre in ascolto, e si occupa di mettere le eventuali richieste ricevute nella lista delle richieste.

I thread dispatcher si occupano di prendere le richieste dall'apposita lista (*popReq*), verificano che la risorsa richiesta non sia in cache: se non lo è si connettono al server, la richiedono e la mettono in cache; in caso contrario vanno a riprendere la richiesta direttamente dalla cache.
Prima di essere inserita in cache, la risorsa viene parsata alla ricerca di ref e idx+ref. Nel caso vengano trovati, le corrispondenti richieste vengono messe nella lista delle richieste (prefetching).

Nel caso in cui il flag *prefetch* contenuto nella struttura request sia 0, significa che la richiesta è arrivata direttamente da un client e quindi va rimandata al client.
Nel caso in cui il flag *prefetch* sia 1 (ref) o 2 (idx+ref) significa che quella richiesta è stata trovata da un parsing della risorsa e quindi non va rimandata al client ma soltanto messa in cache in attesa che venga richiesta dal client.

Prima di inserire la risorsa in cache, controllo che la risposta che mi è arrivata dal server sia completa (tramite il campo *complete* della struttura *response*). In caso negativo, non viene inserita in cache ma inviata comunque al client.

Infine in proxy.c abbiamo implementato la *handleSigTerm* che si occupa di uccidere i thread creati, chiudere i file descriptor aperti e distruggere i mutex e le cond, richiamate quando viene premuto CTRL+C.

Cache.c

Questo file contiene le funzioni per la gestione delle risorse in cache.
La cache è strutturata nel seguente modo: si compone di una lista di *server_elem*, dove ogni server punta all'elemento successivo e ad una lista di risorse contenute in quel

server. La struttura *resource_elem* contiene il puntatore alla risorsa successiva, un'informazione relativa all'istante in cui è stata salvata, e un puntatore alla response corrispondente.
La struttura *response* contiene informazioni riguardante le risposte e un campo booleano *complete:* è TRUE se la risposta è completa (ovvero se il parser ha rilevato che la lunghezza effettiva del blocco è uguale a quella dichiarata in len), FALSE altrimenti.

Funzioni implementate:

- *initCache*: Inizializza i mutex che gestiscono la concorrenza nella Cache.
- *insertServer*: Inserisce un server nella lista dei server cachati. Se il server è gia presente nella lista, viene spostato in testa per motivi di efficenza. Viene ritornato l'indirizzo del server oppure NULL se qualcosa va storto.
- *getResource:* Cerca una risorsa nella lista delle risorse del server specificato, se la trova la restituisce altrimenti restituisce NULL. NULL può anche essere restituito nel caso in cui il tempo della risorsa sia scaduto.
- *insertResource:* Inserisce una risorsa nella lista dei server cachati. Viene passato il flag che indica di che tipo era la richiesta (see also Request.h). Se è di tipo 0 oppure 2 viene parsata alla ricerca ripettivamente di REF e IDX+REF oppure solo di REF. Nel caso in cui vengano trovati, vengono create le richieste corrispondenti che verranno soddisfatte dai dispatcher e messe in cache. Ritorna 1 se l'inserimento va a buon fine, -1 in caso di errore.

IOUtil.c

Contiene le funzioni per lo scambio dei dati tra client, proxy e server.

- *setSockTimeout*: setta l'opzione timeout del fd del socket passato come argomento a INACTIVITY_TIMEOUT_SECONDS
- *setSockReuseAddr*: setta l'opzione reuse address del fd del socket passato come argomento, che permette il riuso immediato dell'indirizzo locale
- *readn*: setta il timeout del socket, e continua a leggere dati finchè la read non ritorna 0. Esce soltanto in caso di EAGAIN (Resource temporarily unavailable).
- *writen*: setta il timeout del socket, e invia dati al destinatario. Riprova finchè la write non ritorna un valore diverso da -1.

List.h

Contiene le inclusioni per la gestione delle liste. Le liste utilizzate e le relative funzioni sono quelle del kernel linux. Si tratta di liste bilinkate e circolari.

Parser.c

Contiene funzioni che si occupano del parsing di richieste e risposte.
Funzioni implementate:

- *parseRequest*: prende in input una stringa e la parsa riempiendo i campi della struttura *request* passata per riferimento come parametro della funzione.
- *stringRequest*: trasforma una struttura *request* in una stringa
- *matchSubstrBool*: date due stringhe, ritorna 1 se la sottostringa compare nella stringa, 0 altrimenti
- *matchSubstr*: date due stringhe, ritorna un puntatore alla fine della sottostringa nella stringa (la prima occorrenza di sub che trova)

- *parseRef*: parsing del blocco ricevuto, alla ricerca di eventuali REF o IDX+REF: vengono salvati nei vettori passati in input. Nel caso il parametro idxRef passato sia NULL, vengono cercati solo i REF contenuti nel blocco.
- *parseResponse*: parsing della risposta: estrae da una risposta l'expire e il blocco, dopo aver verificato che il blocco sia lungo LEN bytes. In caso di errore (risposta non ben formata oppure block incomplete) la funzione ritorna una response con retcode -1

Request.c

Contiene funzioni e strutture dati per la gestione delle richieste. Abbiamo definito la struttura request che contiene il file descriptor relativo al client che l'ha spedita, il tipo, il protocollo, l'IP, la porta, il percorso e un flag prefetch, definito nel seguente modo:

0. È una richiesta che arriva da un client e quindi quando arriva la risposta va spedita al client e messa in cache
1. È una richiesta REF che arriva da una dei dispatcher (prefetching) e quindi va semplicemente messa in cache senza essere rispedita al client
2. È una richiesta IDX+REF che arriva da uno dei dispatcher (prefetching) e quindi va messa in cache e poi parsata per trovare eventuali altre richieste REF, senza essere rispedita al client.

Inoltre abbiamo definito la struttura *request_elem* (di cui è composta la lista delle richieste) che contiene un puntatore alla richiesta successiva e un puntatore alla request corrispondente.

Funzioni implementate:

- *initReq*: Da chiamare per inizializzare le trutture utili alla gestione della coda delle richieste
- *insertReq:* Inserisce una richiesta in coda alla lista delle richieste. Se la lista è piena (*n_req=MAXREQ*), la insert è bloccante return: 1 se l'operazione va a buon fine, 0 altrimenti.
- *popReq:* Estrae una richiesta dalla testa della lista delle richieste. Se la lista è vuota, la pop è bloccante return: la richiesta, oppure NULL se l'operazione fallisce.
- *getServer*: Data una richiesta restituisce una stringa che contiene server:porta che serve ad indetificare il server dentro alla lista dei server.

Scelte implementative

- Per migliorare l'efficienza dell'inserimento in cache, la funzione che si occupa di inserire un server nella lista dei server cachati, lo inserisce in testa se non è gia presenti, altrimenti lo sposta in testa alla lista. La scelta di tenere i server "recenti" in testa è dovuta al fatto che è probabile che verrà presto richiesta un'altra risorsa dallo stesso server.
- Per la gestione della concorrenza, abbiamo scelto di utilizzare pthread per motivi di semplicità e modularità.
- Abbiamo utilizzato le liste del kernel di linux, che sono bilinkate e circolari. Siccome il codice ANSI C non permette l'uso della parola chiave *inline*, abbiamo dovuto definire le funzioni *static* per fa sì che il linker non desse errori di definizioni multiple. Definendo le funzioni *static* abbiamo dovuto utilizzare il flag *-Wno-unused-function* nel Makefile per evitare warning sulle funzioni non utilizzate.