RELAZIONE PROGETTO DI RETI DI CALCOLATORI a.a. 2010/2011
PROF. VITTORIO GHINI

«Proxy miniHTTP/miniHTML con parallelizzazione»

Antonello Antonacci - Ruggero Schirinzi

antonello.antonacci@studio.unibo.it
ruggero.schirinzi@gmail.com

SCOPO DEL PROGETTO:

Utilizzando il protocollo TCP, si deve implementare un proxy che fornisca un servizio di forwarding tra diversi Client ed un Server.
Per ogni richiesta GET ricevuta da un Client, il proxy deve instaurare una connessione con il Server, ottenere informazioni (richiesta INF) sul file voluto dal Client e strutturare una serie di richieste GET da inoltrare al Server, per ottenere il contenuto del file in questione; una volta fatto ciò, il proxy dovrà inoltrare il contenuto del file al Client in attesa.

PANORAMICA GENERALE:

Seguendo le specifiche, il Server utilizzato nel nostro progetto può accettare esclusivamente richieste GET con RANGE. Nel dettaglio, ogni richiesta può specificare un blocco massimo di 10 byte.
Per questo motivo, e sapendo che il Server chiude la connessione con il proxy subito dopo l'invio di ogni singola risposta, il proxy deve prima calcolare il numero necessario di richieste GET e, successivamente, instaurare per ognuna di esse una connessione lato-Server.
Il proxy, implementato utilizzando dei thread, permette il download e l'invio di ogni singola parte del file in parallelo.
Inoltre, è stata implementata una piccola memoria cache, contenente al massimo 20 file, che viene continuamente controllata, ed eventualmente svuotata, da un thread ausiliario chiamato nel corpo del main().

STRUTTURE:

Di seguito, analizzeremo le strutture implementate ed utilizzate nel nostro proxy.

RequestWrapper:

Questa è la struttura principale del proxy, contenente una serie di strutture che verranno utilizzate per il corretto svolgimento delle operazioni.
Il campo 'posizione' viene utilizzato in un array bidimensionale per controllare che, dopo aver ricevuto il blocco dati dal Server, sia effettivamente il proprio turno per inviarli al Client.
All'interno del proxy viene dichiarato un array contenente 100 RequestWrapper, tante quante le connessioni accettate dalla listen() del proxy.

```
struct RequestWrapper {
        struct Request request;
        struct RequestGet request_get;
        struct ResponseGet response_get;
        struct ResponseInf response_inf;
        int posizione;
  };
```

Request:

Questa struttura viene usata nella prima fase della comunicazione lato-Client.
Al suo interno sono presenti due buffer, uno per la ricezione della richiesta ed un altro per l'invio della richiesta INF.
Nel buffer 'PATH' verrà inserito il nome(compresa l'estensione)del file da richiedere.

```
struct Request {
        char buf[MAXLENREQ];
        char inf[MAXLENREQ];/*    buffer per richiesta INF        */
        int lenreq;
        uint16_t port;        /*          porta server          */
        struct in_addr serveraddr; /* ip server  sockaddr_in*/
        char path[MAXLENPATH];
};
```

RequestGet:

Questa struttura viene utilizzata per effettuare le richieste GET con RANGE.
All'interno del campo 'buf' si trova la richiesta vera e propria, impostata secondo il formato delle specifiche.
Nel campo 'maxconnect' viene specificato il numero esatto di connessioni da instaurare con il Server per ottenere l'intero file; questo numero è il valore di ritorno della funzione checkLen.
Il campo 'temp' è l'indice del thread con il quale si sta instaurando la connessione con il Server. Questo valore, oscilla da 0 a maxconnect-1. Viene usato per settare nel modo appropriato i valori della struttura Range.

```
struct RequestGet {
        char buf[MAXLENREQ];
        int maxconnect;
        int temp;//usato per i thread indica il proprio numero
        struct Range range;
        };
```

ResponseInf:

Questa struttura viene riempita quando si deve ricevere la risposta di tipo INF dal Server.
Il messaggio ricevuto viene inserito dentro il campo 'buf'.
Nei due campi successivi avremo i valori di lunghezza e scadenza del file che il Client sta richiedendo.

```
struct ResponseInf {
        char buf[MAXLENREQ];
        int Len;
        int expire;
};
```

ResponseGet:

Questa struttura viene riempita quando si riceve la risposta di tipo GET da parte del Server.
Il messaggio viene inserito all'interno del campo 'buf', mentre in 'fd' abbiamo il file descriptor ottenuto dalla accept() eseguita nel main.
Nel buffer 'dati' vengono inseriti i bytes successivi all'intestazione del messaggio; questo viene fatto perchè solo il primo thread invierà al Client l'intestazione, tutti gli altri si limiteranno ad inviare solo i dati contenuti nel file.
Anche in questo caso è presente una struttura Range; questa però viene usata solo nella funzione check201(), per avere delle variabili di appoggio.

```
struct ResponseGet{
        char buf[MAXLENDATA];
        int fd;//usato per la invio200, qui viene salvato il fd del client
        int expire;
        char dati[MAXLENDATA];
        struct Range range;
        };
```

Life:

Questa struttura viene utilizzata per la gestione della cache.
Il primo campo indica il nome del file che si va ad inserire; questo verrà utilizzato per una rapida ricerca dell'esistenza o meno del file richiesto dal Client nella cache.
Il campo 'buf' contiene il file[nome] comprensivo di intestazione.
Il campo 'birth' contiene la data di inserimento del file[nome] in cache.

```
struct Life{
        char nome[20];
        char buf[5500];
        int expire;
        long int birth;
        };
```

<u>FUNZIONAMENTO GENERALE:</u>

Per ogni connessione accettata lato-Client, il proxy individua il primo posto libero nell' array di strutture RequestWrapper. Dopo di che, passa la struttura ad un thread che ha il compito di gestire l'intera richiesta file.
Il thread attende di ricevere dal Client la richiesta per poi controllarne il formato per un eventuale errore di tipo 403.

A questo punto, il thread controlla se il file richiesto è già presente nella cache; se lo trova, si deve prima controllare che questo non sia scaduto oppure che non abbia subito modifiche dalla data di inserimento nella cache. Se il file è ancora valido, posso direttamente inviarlo al Client, chiudere la connessione lato-Client ed aggiornare il campo birth. In caso contrario il proxy elimina il file scaduto dalla cache e procede normalmente nella gestione della richiesta. Anche nel caso in cui il file non sia presente nella cache, il proxy procede normalmente nella gestione, preoccupandosi di individuare la prima posizione libera utile all´inserimento del file in memoria.

STRUTTURAZIONE RICHIESTE:

Il proxy invia la richiesta di INF al Server (funzione handlerResponseInf) e successivamente dopo aver valutato il campo Len della struttura ResponseInf genererà, attraverso la funzione checkLen(), il numero giusto di thread ognuno dei quali avrà il compito di richiedere una porzione del file tramite apposita richiesta GET.
Ogni thread generato imposterà i giusti campi della struttura Range, creerà una connessione col Server (funzione handlerResponseGet), attenderà di ricevere i dati e, se sarà il suo turno, procederà con l´invio al Client, sbloccando poi il thread successivo; fatto ciò potrà chiudere la propria connessione con il Server.
Terminato l´invio di tutte le parti del file e completata l´attesa attraverso il pthread_join, si potrà chiudere la connessione con il Client.
A questo punto, il thread incaricato della gestione della richiesta libererà la struttura per i successivi Client.
Infine, il file scaricato verrà messo nella cache, se sarà possibile individuare una posizione libera al suo interno.

TEST:

Di seguito è riportato un grafico in cui è possibile vedere il tempo impiegato per completare delle richieste con dei file della dimensione di circa 10, 100, 1040, 2040, 3100 bytes.
Viene messo a confronto in particolare, il tempo necessario a completare l'operazione nel caso in cui il file si trovi già nella cache, oppure debba essere prelevato dal Server.

Il grafico non tiene conto di eventuali errori o ritardi da parte del Server e il contenuto dei file non prevedeva nessun riferimento ad altri file.