Programmazione Distribuita I

Test di programmazione socket, 7 luglio 2016 – Tempo: 2 ore 10 minuti

Il materiale per l'esame si trova nella directory "`exam_dp_jul2016`" all'interno della propria home directory. Per comodità la directory contiene già uno scheletro dei file C che si devono scrivere (nella sotto-directory "`source`"). E' **FORTEMENTE RACCOMANDATO** che il codice venga scritto inserendolo all'interno di questi file **senza spostarli di cartella** e che questo stesso venga letto attentamente e completamente prima di iniziare il lavoro!
L'esame consiste nello scrivere un'altra versione del client e del server dell'esercizio 3.4 dei laboratori del corso (trasferimento di file con XDR). Il nuovo client e server devono usare le seguenti definizioni XDR estese (che sono già disponibili nel file xdr_types.x):

```
enum tagtype {
            GET = 0,
            OK   = 1,
            QUIT   = 2,
            ERR   = 3,
            GETMD5 = 4,
            OKMD5 = 5
};

struct file {
     opaque contents<>;
     unsigned int last_mod_time;
};

union message switch (tagtype tag) {
         case GET:
         case GETMD5:
            string filename<256>;
         case OK:
            struct file fdata;
         case QUIT:
            void;
         case ERR:
            void;
         case OKMD5:
            opaque MD5<16>;
 };
```

Parte 1 (obbligatoria per passare l'esame, max 7 punti)
Scrivere un server (server1) che si comporti secondo il protocollo specificato nell'esercizio di laboratorio 3.4 (che per comodità è incluso alla fine di questo file), ma con le definizioni XDR date sopra. Quando il server1 riceve un messaggio GETMD5, il server1 deve aprire il file specificato e calcolarne l'hash MD5 del contenuto. Se il file può essere aperto e letto senza errori, il server1 risponde con il messaggio OKMD5 che include il valore MD5 calcolato. Se invece il file non può essere aperto o letto, il server1 risponde con il messaggio ERR. Per il

calcolo dell'MD5 del file, il server1 deve usare le funzioni di utilità contenute in md5.c di cui un esempio di uso è fornito nel file utility.c. Si leggano attentamente i commenti presenti in utility.c.
Il server deve ascoltare sulla porta specificata come primo argomento, su tutte le interfacce di rete disponibili.
Il/i files C del programma server devono essere scritti dentro una directory `$ROOT/source/server1`, dove `$ROOT` è la directory dell'esame (ovvero la cartella "exam_dp_jul2016"). Se si devono includere files di libreria (per esempio quelli dello Stevens) che si vogliono riusare per le parti seguenti, questi files possono essere messi nella directory `$ROOT/source` (si ricorda che per includere questi files nei propri sorgenti è necessario specificare il percorso "`..`").
Per testare il proprio server si può lanciare il comando

```
./test.sh
```

dalla directory `$ROOT` . Si noti che il programma lancia anche altri tests (per le parti successive). Il programma indicherà se almeno i test obbligatori sono passati.

Parte 2 (max 4 punti)
Scrivere un client (chiamato `client`) si comporta come quello sviluppato nel laboratorio 3.4, ma che usi le definizioni XDR date sopra.
Il nuovo client deve usare i seguenti parametri dalla linea di comando. La lista degli argomenti può iniziare con il parametro opzionale "-x" (se presente, il client deve usare il protocollo basato su XDR), seguito (nell'ordine esattamente come specificato di seguito) dall'indirizzo IPv4 o l'hostname del server, dal numero di porta del server espresso come numero decimale, dal nome di un file da scaricare dal server o di cui richiedere l'MD5 al server. Quando il "-x" è presente come primo parametro, è possibile avere, come quinto parametro opzionale, "-md5". Se questo parametro è presente, il client deve ricevere dal server l'hash MD5 del file specificato, mentre se l'argomento è assente il client deve scaricarne il contenuto.
L'indirizzo IP del server è espresso in notazione standard (decimale puntata), mentre la porta è espressa come un numero decimale.
Il file scaricato dal client deve essere salvato nella directory di lavoro corrente del client, con lo stesso nome del file originale, mentre l'hash MD5 deve essere salvato nella directory di lavoro corrente del client con lo stesso nome del file ma con suffisso ".md5". Per esempio, se il file da richiedere si chiama "f.txt", il contenuto dell'MD5 del file deve essere salvato dal client in un file di nome "f.txt.md5".
L'hash MD5 deve essere scritto in formato testo esadecimale, un byte di seguito all'altro, senza spazi, utilizzando il formattatore "%02x" della printf, e terminando con "\n". (Il formato testo esadecimale è lo stesso stampato dal comando md5sum).
Il/i files C del programma client devono essere scritti dentro una directory `$ROOT/source/client`, dove `$ROOT` è la directory dell'esame (ovvero la cartella "exam_dp_jul2016"). Il comando di test indicato per la Parte 1 tenterà di testare anche la Parte 2.

Parte 3 (max 5 punti)
Scrivere una nuova versione del server sviluppato nella prima parte (server2) che sia in grado di servire almeno 3 clients in parallelo (per il server1 non c'era alcun requisito riguardo la concorrenza). In aggiunta, il server2 deve gestire un parametro addizionale sulla linea di comando. Questo parametro (chiamato "n") deve essere un numero intero. Se n è 0, il server2 deve creare un processo figlio per ogni richiesta, e quando questo ha terminato di servire la richiesta, il figlio deve terminare. Se invece n è maggiore di 0 e non più grande di 10, il server2 deve usare la tecnica della pre-allocazione (pre-forking) ed n rappresenta il numero di processi che possono servire le richieste dal client.
Il/i files C del programma server2 devono essere scritti dentro una directory `$ROOT/source/server2`, dove `$ROOT` è la directory dell'esame (ovvero la cartella "exam_dp_jul2016"). Il comando di test indicato per la Parte 1 tenterà di testare anche il server2.

Ulteriori istruzioni (comuni a tutte le parti)
Per passare l'esame è necessario almeno implementare un server1 che invii un hash MD5 codificato correttamente al client, oppure è necessario implementare un server1 ed un client che possano interagire tra loro come specificato.
La soluzione sarà considerata valida **se e solo se** può essere compilata tramite i seguenti comandi lanciati dalla cartella `source` (gli scheletri forniti compilano già tramite questi comandi, non spostarli, riempirli solo):

```
rpcgen -h -o xdr_types.h xdr_types.x

rpcgen -c -o xdr_types.c xdr_types.x

gcc -o socket_server1 server1/*.c *.c -Iserver1 -lpthread -lm

gcc -o socket_client client/*.c *.c -Iclient -lpthread -lm

gcc -o socket_server2 server2/*.c *.c -Iserver2 -lpthread -lm
```

Si noti che tutti i files che sono necessari alla compilazione del client e del server devono essere inclusi nella directory `source` (per esempio è possibile usare i files dal libro dello Stevens, ma questi files devono essere inclusi da chi sviluppa la soluzione).
Per comodità, il programma di test controlla anche che la soluzione possa essere compilata con i comandi precedenti.
Tutti i files sorgenti prodotti (`server1, server2, client` e i files comuni) devono essere inclusi in un singolo archivio zip creato tramite il seguente comando bash (lanciato dalla cartella `exam_dp_jul2016`):

```
./makezip.sh
```

Il file zip con la soluzione deve essere lasciato nella directory in cui è stato creato dal comando precedente.

Nota: controllare che il file zip sia stato creato correttamente estraendone il contenuto in una directory vuota, controllando il contenuto, e controllando che i comandi di compilazioni funzionino con successo (o che il programma di test funzioni).

Attenzione: gli ultimi 10 minuti dell'esame DEVONO essere usati per preparare l'archivio zip e per controllarlo (e aggiustare eventuali problemi). Se non sarà possibile produrre un file zip valido negli ultimi 10 minuti l'esame verrà considerato non superato.

La valutazione del lavoro sarà basata sui tests forniti ma anche altri aspetti del programma consegnato (per esempio la robustezza) saranno valutati. Di conseguenza, passare tutti i tests non significa che si otterrà necessariamente il punteggio massimo. Durante lo sviluppo del codice si faccia attenzione a scrivere un buon programma, non solo un programma che passa i tests forniti.

Esercizio 3.4 (XDR-based file transfer)

Modificare il client ed il server sviluppati fino a questo punto per il trasferimento di file in modo che accettino un parametro opzionale -x prima degli altri argomenti (cioè come primo argomento). Se questo argomento è presente sulla linea di comando, un protocollo simile al precedente, ma basato su XDR, deve essere usato. Secondo questo nuovo protocollo, i messaggi dal client al server e viceversa sono tutti rappresentati tramite il tipo XDR “message” definito come segue:

```
enum tagtype {
            GET = 0,
            OK   = 1,
            QUIT   = 2,
            ERR   = 3
};

struct file {
        opaque contents<>;
        unsigned int last_mod_time;
};

union message switch (tagtype tag) {
         case GET:
            string filename<256>;
         case OK:
            struct file fdata;
         case QUIT:
            void;
         case ERR:
            void;
};
```