Programmazione Distribuita I

Test di programmazione socket, 21 luglio 2015 – Tempo: 2 ore

Il materiale per l'esame si trova nella directory "`exam_dp_jul2015`" 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 2.3 dei laboratori del corso (trasferimento di file) che una un protocollo leggermente differente. Per passare l'esame è necessario almeno scrivere il server che si comporti secondo le specifiche della parte 1.

Parte 1 (obbligatoria per passare l'esame, max 8 punti)
Scrivere un server (server1) che si comporti secondo il protocollo specificato nell'esercizio di laboratorio 2.3, ma che usi un formato differente per i messaggi, basati su XDR. La specifica XDR che deve essere usata si trova nel file `types.x`, che è disponibile nella cartella `source` del materiale dell'esame. I messaggi dal client al server (GET e QUIT) sono messaggi XDR specificati dal tipo XDR chiamato `call_msg`, mentre i messaggi dal server al client sono composti da due parti. La prima parte è un messaggio XDR specificato dal tipo XDR chiamato `response_msg`. La seconda parte, che è presente solo se la prima parte è OK, ha esattamente lo stesso formato usato nel laboratorio 2.3, ossia un intero segna segno su 32 bit in network byte order che specifica il numero di bytes del file da trasferire, seguito dai bytes del file.

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_jul2015"). Se si devono includere file di libreria (per esempio quello 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 "`../source`"). La generazione del codice a partire da `types.x` deve essere effettuata come specificato nella parte "Ulteriori istruzioni" (comune a tutte le parti).
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.
Si noti che per passare l'esame è sufficiente implementare un server1 che risponda ad un messaggio GET spedendo un messaggio di risposta codificato correttamente.

Parte 2 (max 6 punti)
Scrivere un client (chiamato `client`) si comporta come quello sviluppato nel laboratorio 2.3, ma con le seguenti differenze:

1. Il nuovo client deve usare il formato di messaggi basato su XDR specificato nella Parte 1.
2. Il nuovo client deve usare i seguenti parametri dalla linea di comando: il primo argomento è l'indirizzo IPv4 del server, il secondo argomento è il numero di porta del server, il terzo argomento è il nome di un file che deve essere scaricato dal server.
3. Dopo aver terminato il download del file specificato come terzo parametro della linea di comando, il nuovo client deve spedire il comando QUIT al server e terminare.

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_jul2015").

Parte 3 (max 2 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). Se il server1 aveva già questa funzionalità, si può semplicemente copiare il codice di server1 su server2.

Il/i files C del programma server devono essere scritti dentro una directory `$ROOT/source/server2`, dove `$ROOT` è la directory dell'esame (ovvero la cartella "exam_dp_jul2015"). Il comando di test indicato per la Parte 1 tenterà di testare anche il server2.

Ulteriori istruzioni (comuni a tutte le parti)

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 types.x -o types.h

rpcgen -c types.x -o types.c

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_jul2015`):

```
zip  socket.zip  source/client/*.[ch]  source/server[12]/*.[ch]  source/*.[ch]
```

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 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.