Programmazione Distribuita I

Test di programmazione socket, 3 febbraio 2015 – Tempo: 2 ore 10 minuti

Il materiale per l'esame si trova nella directory "`exam_dp_feb2015`" 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 client e un server che testino un trasferimento dati su una connessione TCP/IP, secondo le specifiche descritte nel seguito. Per passare l'esame è necessario almeno scrivere il server che si comporti secondo le specifiche di seguito.

Parte 1 (obbligatoria per passare l'esame, max 8 punti)
Scrivere un server (server1) che riceve due argomenti sulla linea di comando. Il primo argomento è il numero intero decimale della porta TCP sulla quale il server sta in ascolto, mentre il secondo è un altro numero intero decimale che specifica quanti Megabytes (cioè 2^{20} bytes = 1048576 byte) il server deve ricevere su ognuna delle connessioni che vengono stabilite. Nel momento in cui una connessione viene stabilita, il server inizia a ricevere dati dal client sulla connessione. Mentre riceve i dati, il server calcola un hash su 32-bit dei bytes ricevuti (vedere oltre per come fare). Quanto il numero totale di Megabytes specificati come secondo argomento sulla linea di comando sono stati ricevuti, il server invia l'hash su 32-bit (4 bytes) al client in network byte order, stampa l'hash come numero intero decimale sullo standard output e poi chiude ordinatamente la connessione.
L'hash su 32-bit si deve calcolare tramite la funzione `hashCode` il cui codice sorgente è incluso nel file `utils.c` (nella cartella `source`). Per calcolare l'hash lavorando sui dati segmento per segmento, si proceda nel modo seguente: chiamare `hashCode(data,n,1)`, dove `data` è il primo segmento di dati e `n` è la sua lunghezza. Successivamente, per ogni segmento di dati successivo, si chiami `hashCode(data,n,hc)`, dove `data` e `n` sono il nuovo segmento di dati e la sua lunghezza, mentre `hc` è il risultato ottenuto dalla precedente chiamata di `hashCode`. L'intero ritornato dall'ultima chiamata di `hashCode` è l'hash che deve essere spedito dal server.
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_feb2015"). 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`").
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) che riceve esattamente 4 argomenti sulla linea di comando: il primo argomento è l'indirizzo IP del server in notazione decimale puntata, il secondo argomento è il numero intero decimale della porta TCP, il terzo è il numero intero decimale che specifica quanti Megabytes il client deve spedire e l'ultimo argomento è il nome di un file locale.
All'avvio il client deve riempire un buffer in memoria di 1 MegaByte con il primo Megabyte di dati letti dal file locale il cui nome è specificato come l'ultimo argomento sulla linea di comando. Se il file non esiste o non contiene almeno 1 Megabyte, il client deve terminare con codice di uscita (exit code) 1. Poi il client deve continuare a spedire il contenuto del buffer di memoria da 1 Megabyte per n volte (dove n è l'intero specificato come terzo argomento sulla linea di comando), cioè n copie del buffer devono essere spedite al server. Infine, il client deve ricevere l'hash spedito dal server (4 bytes in network byte order), stamparlo come numero intero decimale sullo standard output 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_feb2015").

Parte 3 (max 4 punti)
Scrivere una nuova versione del server sviluppato nella Parte 1 (server2), seguendo le specifiche addizionali qui di seguito elencate.
Il server deve essere in grado di servire almeno 3 clients in parallelo (per il server1 non c'era alcun requisito riguardo la concorrenza).
Il server deve essere in grado di rilevare interruzioni inaspettate del flusso di dati dal client: se il server sta ancora attendendo dati ma non vengono ricevuti ulteriori dati per un periodo di inattività di 10 secondi il server deve chiudere la connessione.
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_feb2015"). 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):

```
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_feb2015`):

```
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.