Programmazione Distribuita I

Test di programmazione socket, 14 settembre 2016 – Tempo: 2 ore 10 minuti

Il materiale per l'esame si trova nella directory "`exam_dp_sep2016`" 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 una versione differente del client e del server sviluppati nell'esercizio di laboratorio 1.4 (client-server UDP base, il testo dell'esercizio di laboratorio è disponibile alla fine di questo documento per comodità). Il nuovo client e server devono usare il seguente protocollo per la comunicazione:

- La richiesta (spedita dal client) è un datagram UDP che contiene una sequenza di caratteri ASCII, da interpretarsi come un nome di file, per il quale il client vuole alcune informazioni (se un file con quel nome è disponibile sul server e qual è la sua dimensione in bytes). Questa sequenza di caratteri non ha un carattere di terminazione (la sua fine è quella del datagram), e la sua lunghezza massima è 256 caratteri. Solo i caratteri alfanumerici, l'underscore e il punto sono permessi come parte del nome del file.
- La riposta (spedita dal server quando riceve una richiesta) è un datagram UDP che contiene la seguente sequenza di campi (esattamente in quest'ordine):
    - un codice intero di 4 byte, rappresentato in network byte order, che può assumere i seguenti valori:
        - 0 nel caso di errore nella richiesta (richiesta troppo lunga o presenza di caratteri non consentiti)
        - 1 nel caso il file richiesto non esista nella directory corrente del server
        - 2 nel caso il file richiesto esista nella directory corrente del server
    - una copia della sequenza di caratteri ASCII spediti dal client nella richiesta (limitati ai primi 256 caratteri se la richiesta eccede questo limite)
    - un altro intero di 4 byte, rappresentato in network byte order, che deve essere incluso solamente se il codice del primo intero di 4 byte è 2, e il cui valore è la lunghezza del file, in bytes, con il nome richiesto.
- Si noti che il secondo campo (copia della sequenza di caratteri ASCII) non ha delimitazioni. La sua lunghezza può essere ottenuta tramite differenza (lunghezza del datagram meno 4 o meno 8, secondo se il terzo campo è presente oppure no).
- Il client devono ignorare (senza ulteriori azioni) messaggi di risposta che hanno codifica errata o che non coincidono con il nome del file che il client ha richiesto.

Parte 1 (obbligatoria per passare l'esame, max 7 punti)
Scrivere un server (server1) che si comporti secondo il protocollo specificato in precedenza. Il server deve ascoltare sulla porta specificata come primo (e unico) argomento, su tutte le interfacce di rete disponibili.
Ogni volta che il server riceve una richiesta da un client, prima di spedire la risposta, deve stampare sullo standard output una linea di testo terminata **da newline (\n)** con il seguente formato:

<prefix> <client-ip> <client-port> <codice-di-risposta> <filename>

dove:

- <prefix> è una stringa ASCII fissa di 7 caratteri “REQUEST”
- <client-ip> è l’indirizzo IP del client che ha inviato la richiesta, in notazione decimale puntata
- <client-port> è il numero di porta, stampato come numero decimale
- <codice-di-risposta> è il codice di risposta che il serve include nella risposta, stampato come numero decimale
- <filename> è il nome del file richiesto dal client (una sequenza di caratteri ASCII), da omettere in caso di errore nella richiesta (es. una richiesta che produce un codice di risposta 0)

Ognuno dei campi adiacenti in questa linea di testo deve essere separato da un solo spazio e ogni linea deve essere terminata, come di consueto, con il carattere \n.
Importante: queste linee di testo devono essere il solo output scritto dal server1 sollo standard output.
Importante: dopo che l’output è stato prodotto, se ne faccia il flush.
Suggerimento: la dimensione del file può essere ottenuta chiamando la funzione stat(). Vedere la pagina di manuale (man stat) per i dettagli.

Il/i files C del programma server devono essere scritti dentro una directory `$ROOT/source/server1`, dove `$ROOT` è la directory dell’esame (exam_dp_sep2016). 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 5 punti)
Scrivere un client (chiamato `client`) che si comporta come quello sviluppato nel laboratorio 2.1 (il testo dell’esercizio di laboratorio è disponibile alla fine di questo documento per comodità), ma che usi il protocollo specificato sopra. A differenza del lab 2.1, questo client deve tentare una singola ritrasmissione prima di terminare; il tempo di timeout è 3 secondi, così come nell’esercizio di laboratorio 2.1. Nel caso in cui il client riceve una risposta che non è formattata correttamente o che non coincide con il nome file richiesto, il client deve ignorarla senza ulteriori azioni e aspettare per ulteriori 3 secondi.
Il nuovo client deve ricevere i seguenti parametri dalla linea di comando (esattamente nell’ordine qui specificato): l’indirizzo IPv4 o l’hostname del server, il numero di porta del server, e il nome del file da inviare al server. L’indirizzo IP del server è espresso in notazione standard (decimale puntata), mentre la porta è espressa come un numero decimale.
Il client deve stampare sullo standard output una linea di testo terminata **da newline (\n)** con il seguente formato:

<prefix> <response-code> <size-of-file>

dove:

- <prefix> è una stringa ASCII fissa di 8 caratteri “RESPONSE”
- <response-code> è il codice di risposta ricevuto come primo campo della risposta oppure -1 nel caso in cui nessuna risposta sia stata ricevuta. Questo numero è espresso in notazione decimale.
- <size-of-file> è la dimensione del file in bytes (assente se non è stata ricevuta dal server)

Ognuno dei campi adiacenti in questa linea di testo deve essere separato da un solo spazio e ogni linea deve essere terminata, come di consueto, con il carattere \n.
Importante: queste linee di testo devono essere il solo output scritto dal server1 sollo standard output.
Importante: dopo che l’output è stato prodotto, se ne faccia il flush.

Il/i files C del programma client devono essere scritti dentro una directory `$ROOT/source/client`, dove `$ROOT` è la directory dell’esame (exam_dp_sep2016). Il comando di test indicato per la Parte 1 tenterà di testare anche la Parte 2.

Parte 3 (max 4 punti)
Scrivere una nuova versione del server sviluppato nella prima parte (server2) che sia in grado di servire clients in maniera concorrente usando il pre-forking (per il server1 non c’era alcun requisito riguardo la concorrenza). Il server deve ricevere il numero di processi che possono servire le richieste dei client in maniera concorrente come secondo argomento sulla linea di comando (dopo il numero di porta).
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_sep2016”). 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 una risposta codifica correttamente al client, oppure è necessario implementare un server1 ed un client che possano interagire tra loro come ci si aspetta.
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_sep2016`):

```
./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 1.4 (client-server UDP base)

Scrivere un client che invii ad un server UDP (all'indirizzo e porta specificati come primo e secondo parametro sulla riga di comando) un datagramma contenente il nome (massimo 31 caratteri) passato come terzo parametro sulla riga di comando, attenda quindi un qualunque datagramma di risposta dal server. Il client termina mostrando il contenuto (testo ASCII) del datagramma ricevuto oppure segnalando che non ha ricevuto risposta dal server entro un certo tempo.

Sviluppare un server UDP (in ascolto sulla porta specificata come primo parametro sulla riga di comando) che risponda ad ogni datagramma ricevuto inviando un datagramma di risposta contenente lo stesso nome presente nel pacchetto ricevuto.

Provare ad effettuare degli scambi di datagrammi tra client e server con le seguenti configurazioni:

- client invia datagramma ad una porta su cui il server è in ascolto
- client invia datagramma ad una porta su cui il server non è in ascolto
- client invia datagramma ad un indirizzo non raggiungibile (es. 10.0.0.1)

Nel materiale del laboratorio sono disponibili le versioni eseguibili del client e del server che si comportano secondo le specifiche fornite. Provare a collegare il proprio client col server fornito nel materiale del laboratorio, e il client fornito nel materiale con il proprio server. Se sono necessarie correzioni fare in modo che il proprio server e il proprio client continuino a comunicare correttamente con il client e il server fornito. Per terminare correttamente l'esercizio è necessario arrivare ad avere un client ed un server che comunicano correttamente tra loro e che possono comunicare correttamente con il client ed il server forniti nel materiale del laboratorio.

Esercizio 2.1 (client UDP perseverante)

Modificare il client UDP dell'esercizio 1.4 in modo che – se non riceve qualsiasi risposta dal server entro 3 secondi – ritrasmetta la richiesta (fino ad un massimo di 5 volte) e quindi termini indicando se ha ricevuto risposta o meno.

Effettuare le stesse prove dell'esercizio 1.4.