Programmazione Distribuita I

Test di programmazione socket, 22 giugno 2016 – Tempo: 2 ore 10 minuti

Il materiale per l'esame si trova nella directory "`exam_dp_jun2016`" 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.

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 2.3 (che per comodità è incluso alla fine di questo file), ma con una piccola variazione: nel momento in cui il server finisce di leggere il comando dal client, il server calcola il timestamp TS1 (usando la funzione gettimeofday). Dopo aver spedito il file richiesto, il server calcola un altro timestamp TS2, e calcola anche la differenza TS2-TS1 in microsecondi. Questo numero è codificato come numero intero su 32 bit senza segno in network byte order ed è spedito immediatamente dopo il contenuto del file.
Il server deve ascoltare sulla porta specificata come primo argomento, su tutte le interfacce di rete disponibili.
Dopo aver spedito la risposta al client nel modo specificato sopra, il server deve stampare una sola linea di testo sullo standard output con il seguente formato:

<prefix> <client-ip> <client-port> <filename> <TS2-TS1>

dove:

- <prefix> è una stringa ASCII fissa di 9 caratteri "OPERATION"
- <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
- <filename> è il nome del file sotto forma di sequenza di caratteri ASCII
- <TS2-TS1> è lo stesso valore spedito al client, stampato come numero decimale

Ognuno dei campi adiacenti in questa linea di testo deve essere separato da un solo spazio.
Importante: queste linee di testo devono essere il solo output scritto dal server1 sollo standard output.

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_jun2016"). 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 2.3, ma con le seguenti differenze:

1. Il nuovo client deve usare i seguenti parametri dalla linea di comando: il primo argomento è l'indirizzo IP (IPv4 o IPv6) o l'hostname del server, il secondo argomento e il numero di porta del server espresso come numero decimale, il terzo argomento è il nome di un file da scaricare dal server. Se il primo argomento rappresenta l'indirizzo IP del server, questo è espresso in notazione standard (decimale o esadecimale), mentre la porta è espressa come un numero decimale.
2. Il client deve collegarsi al server specificato sulla linea di comando e scaricare il file specificato sulla linea di comando stessa usando il protocollo modificato descritto nella Parte 1. Dopo questa operazione, il client deve stampare il valore TS2-TS1 ricevuto dal server come numero decimale, preceduto dalla stringa di 9 caratteri "TIMESTAMP", e separato da quest'ultima con un solo spazio, e terminare. Il numero decimale deve essere il **solo** output del client sullo standard output.

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_jun2016"). 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 chiudere in maniera corretta la connessione con il client se nessun comando è ricevuto entro 10 secondi dall'inizio della connessione o da quando l'ultimo trasferimento file è stato completato.
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_jun2016"). 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 risponda al client con un file codificato correttamente seguito dall'informazione di timestamp specificata, e che stampi l'informazione prevista sullo standard output, 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):

```
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_jun2016`):

```
./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 2.3 (server TCP iterativo)

Sviluppare un server TCP (in ascolto sulla porta specificata come primo parametro sulla riga di comando) che accetti richieste di trasferimento file da client ed invii il file richiesto.

Sviluppare un client che possa collegarsi ad un server TCP (all'indirizzo e porta specificati come primo e secondo parametro sulla riga di comando) per richiedere dei file e memorizzarli localmente. I nomi dei file da richiedere vengono forniti su standard input, uno per riga. Ogni file richiesto deve essere salvato localmente e deve essere stampato su standard output un messaggio circa l'avvenuto trasferimento, con nome, dimensione del file e timestamp di ultima modifica.

Il protocollo per il trasferimento del file funziona come segue: per richiedere un file il client invia al server i tre caratteri ASCII "GET" seguito dal carattere ASCII dello spazio e dai caratteri ASCII del nome del file, terminati da CR LF (*carriage return* e *line feed*, cioè due bytes corrispondenti ai valori 0x0d 0x0a in esadecimale, ossia '\r' e '\n' in notazione C, sempre senza spazi):

G	E	T		…filename…	CR	LF

(Nota: il comando include un totale di 6 caratteri più quelli del nome del file)

Il server risponde inviando:

+	O	K	CR	LF	B1	B2	B3	B4	T1	T2	T3	T4	File content……

Notare che il messaggio è composto da 5 caratteri, seguiti dal numero di byte del file richiesto (un intero senza segno su 32 bit in network byte order - bytes B1 B2 B3 B4 nella figura), e quindi dal timestamp dell'ultima modifica (Unix time, cioè numero di secondi dall'inizio dell' "epoca"), rappresentato come un intero senza segno su 32 bit in network byte order (bytes T1 T2 T3 T4 nella figura), e infine dai byte del file in oggetto.

Per ottenere il timestamp dell'ultima modifica al file, si faccia riferimento alle chiamate di sistema *stat* o *fstat*.

Il client può richiedere più file inviando più comandi GET. Quando intende terminare la comunicazione invia:

Q	U	I	T	CR	LF

(6 caratteri) e chiude il canale.

In caso di errore (es. comando illegale, file inesistente) il server risponde sempre con

-	E	R	R	CR	LF

(6 caratteri) e quindi chiude il canale col client.

Si faccia attenzione a porre l'eseguibile del client in una directory DIVERSA da quella dove gira il server. Si chiami questa cartella con lo stesso nome del programma client (questo per evitare che, con prove sullo stesso PC, salvando il file si sovrascriva lo stesso file che viene letto dal server).

Provare a collegare il proprio client con il server incluso nel materiale fornito per il laboratorio, ed il client incluso nel materiale fornito con il proprio server (notare che i files eseguibili sono forniti per architetture sia a 32 bit sia a 64 bit. I files con suffisso _32 sono compilato per girare su sistemi Linux a 32 bit, quelli senza suffisso per sistemi a 64 bit. I computer al LABINF sono sistemi a 64 bit). Se sono necessarie delle modifiche al proprio client o al server, controllare attentamente che il client ed il server modificati comunichino correttamente sia tra loro sia con i client e server forniti. Al termine dell'esercizio, si deve avere un client e un server che possono comunicare tra loro e possono operare correttamente con il client ed il server forniti nel materiale del laboratorio.

Provare a trasferire un file binario di dimensioni notevoli (circa 100 MB). Verificare che il file sia identico tramite il comando cmp o diff e che l'implementazione sviluppata sia efficiente nel trasferire il file in termini di tempo di scaricamento.

Mentre è in corso un collegamento provare ad attivare un secondo client verso il medesimo server.

Provare ad attivare sul medesimo nodo una seconda istanza del server sulla medesima porta.

Provare a collegare il client ad un indirizzo esistente ma ad una porta su cui il server non è in ascolto.

Provare a disattivare il server (battendo CTRL+C nella sua finestra) mentre un client è collegato.