Programmazione Distribuita I

Test di programmazione socket, 28 giugno 2017 – Tempo: 2 ore 10 minuti

Il materiale per l'esame si trova nella directory "`exam_dp_jun2017`" all'interno della propria home directory, che verrà chiamata `$ROOT` nel seguito.
Per comodità la directory contiene già uno scheletro dei file C che si devono scrivere (nella directory "`$ROOT/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 versioni differenti del client e del server sviluppati nell'esercizio di laboratorio 2.3 (trasferimento file), che usano un protocollo leggermente differente. Secondo il nuovo protocollo, viene utilizzata una codifica di dati in formato binario per i messaggi, invece che la codifica ASCII specificata nell'esercizio 2.3:

- Il messaggio GET è costituito da 1 byte, impostato al valore intero 0, seguito da un intero senza segno su 2 bytes in network byte order (L1 L2) che rappresenta la lunghezza del nome del file, seguito dai caratteri ASCII del nome del file.:

0	L1	L2	…nome file…

- La risposta positiva del server (cioè il messaggio +OK dell'esercizio 2.3) è costituito da 1 byte impostato al valore intero 1, seguito da due interi senza segno su 4-bytes (B1 B2 B3 B4 and T1 T2 T3 T4), seguiti dal contenuto del file:

1	B1	B2	B3	B4	T1	T2	T3	T4	Contenuto file……

  B1 B2 B3 B4 and T1 T2 T3 T4 hanno lo stesso significato e codifica del protocollo originale.
- Il messaggio QUIT è costituito da 1 byte impostato al valore intero 2.
- La risposta negativa del server (cioè il messaggio –ERR dell'esercizio 2.3) è costituito da 1 byte impostato al valore intero 3.

Parte 1 (obbligatoria per passare l'esame, max 6 punti)
Copiare il proprio server che è parte della soluzione dell'esercizio di laboratorio 2.3 nel template C del main del client sotto la directory `$ROOT/source/server1`, e modificarlo così che funzioni secondo la nuova versione del protocollo specificata prima nel testo.
Il server deve ascoltare sulla porta specificata come unico argomento della linea di comando (come numero decimale), su tutte le interfacce disponibili.
Il/i files C del programma server devono essere tutti scritti dentro la directory `$ROOT/source/server1`, ad eccezione dei file comuni (per esempio quelli dello Stevens) che si vogliono riusare negli altri programmi da sviluppare per questo esame, e che 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 client 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 6 punti)
Copiare il client che è parte della propria soluzione del laboratorio 2.3 nel template C del main del client sotto la directory `$ROOT/source/client1`, e modificarlo così che funzioni secondo la nuova versione del protocollo specificata prima nel testo. Se necessario, modificare il proprio client così che riceva l'indirizzo IPv4 (in notazione decimale puntata) e il numero della porta (in notazione decimale) del server a cui collegarsi rispettivamente come primo e secondo argomento della linea di comando, e il nome del file da scaricare come terzo argomento. Si noti che per semplicità questo client non deve richiedere una lista di files come nel laboratorio 2.3; invece, deve scaricare un singolo file il cui nome è specificato come terzo parametro della linea di comando, quindi spedire il comando QUIT, poi chiudere in modo ordinato la connessione, e terminare.
Il/i files C del programma client devono essere scritti tutti dentro la directory `$ROOT/source/client1`.
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 Parte 1 (qui chiamato `server2`) che si comporti come il `server1` ma che

- sia un server concorrente che utilizza il pre-forking, e che può servire fino a 3 clients contemporaneamente (per `server1` non c'era alcun requisito riguardo la concorrenza).
- se una connessione è stabilita ma nessun commando è ricevuto entro 10 secondi, il server deve chiudere la connessione.
- Similmente, dopo aver finito di spedire il messaggio di risposta positiva, incluso il trasferimento di file, il server deve attendere un altro messaggio per al più 10 secondi, e chiudere la connessione se nessun messaggio arriva entro tale tempo.

Il/i files C del programma server2 devono essere scritti dentro una directory `$ROOT/source/server2`. Il comando di test indicato per la Parte 1 tenterà di testare anche la Parte 3.

Ulteriori istruzioni (comuni a tutte le parti)
Per passare l'esame è necessario almeno implementare un `server1` che risponda correttamente ad una richiesta di file secondo il nuovo protocollo, oppure è necessario implementare un `client1` ed un `server1` che possano interagire tra loro come ci si aspetta (cioè, che siano almeno in grado di scambiarsi correttamente un piccolo file).
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_client1 client1/*.c *.c -Iclient1 -lpthread -lm
```

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

Per comodità, il programma di test controlla anche che la soluzione possa essere compilata con i comandi precedenti.
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).
Tutti i files sorgenti prodotti (`client1`, `server1`, `server2`,e i files comuni) devono essere inclusi in un singolo archivio zip creato tramite il seguente comando bash (lanciato dalla cartella `$ROOT`):

```
./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.

Per comodità, il testo dell'esercizio di laboratorio 2.3 è riportato nel seguito.

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.