Programmazione Distribuita I

Test di programmazione socket, 9 settembre 2019 – Tempo: 2 ore 10 minuti

Il materiale per l'esame si trova nella directory "`exam_dp_sep2019`" 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 un'altra versione del client e del server sviluppati per l'esercizio di laboratorio 2.3 (trasferimento files), che usino una versione del protocollo leggermente differente. Secondo il protocollo modificato, la dimensione del file non è più rappresentata in formato binario, ma come stringa di caratteri ASCII numerici che rappresentano il numero di bytes del file in notazione decimale. Questa stringa è terminata da una coppia di caratteri CR LF:

+	O	K	CR	LF	Dimensione file (ASCII)	CR	LF	Contenuto file ……	T1	T2	T3	T4

Parte 1 (obbligatoria per passare l'esame, max 5 punti)
Copiare il server, parte della propria soluzione al laboratorio 2.3 (server TCP iterativo), nel template in linguaggio C che si trova nella directory `$ROOT/source/server1`, e modificarlo così che funzioni secondo la nuova versione modificata del protocollo specificata in precedenza.
A parte quanto specificato sopra, tutte le altre caratteristiche del server1 sono le stesse di quelle della versione originale dell'esercizio.
Il/i files C del nuovo programma server devono essere tutti scritti dentro la directory `$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 devono 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)
Copiare il proprio client, parte della propria soluzione al laboratorio 2.3, nel template in linguaggio C che si trova nella directory `$ROOT/source/client1`, e modificarlo in modo che possa funzionare usando la nuova versione del protocollo specificata in precedenza.
Se il client riceve dal server una risposta con una codifica della dimensione del file sbagliata (per esempio che include caratteri non numerici), deve terminare con un messaggio di errore.
A parte queste specifiche, tutte le altre caratteristiche del client1 sono le stesse della versione originale.
Il/i files C del programma client devono essere tutti scritti dentro la directory `$ROOT/source/client1`, ad eccezione dei file comuni che, come già detto, devono essere messi nella directory `$ROOT/source`.

Il comando di test indicato per la Parte 1 tenterà di testare anche la Parte 2.

Parte 3 (max 6 punti)
Scrivere una nuova versione del server sviluppato nella Parte 1 (qui chiamato `server2`) che si comporti come il `server1` ma che serva più client contemporaneamente usando la creazione di processi al bisogno.
Questo server deve anche scrivere un file di log (nella sua directory di lavoro), chiamato `connections.log`
Il server deve scrivere una linea nel file di log per ogni connessione stabilita con un client. Il formato della linea deve essere come segue:
`Connection accepted from` <*client-ip*> <*client-port*>
dove:

- <*client-ip*> è l'indirizzo IP del client che ha fatto la richiesta, in notazione decimale puntata
- <*client-port*> è la porta del client scritta come numero decimale

Questi campi devono essere separati da UNO spazio nella linea di testo.
Se non sono stabilite nuove connessioni per n secondi (dove n è il valore di un parametro addizionale sulla linea di comando del server, rappresentato come numero decimale positivo), il server deve scrivere la linea seguente nel file di log:
`No new connections for` <*n*> `seconds`
dove <*n*> è il parametro addizionale sulla linea di comando del server.
Questa linea di testo deve essere scritta ripetutamente ogni n secondi mentre continua l'assenza di connessioni. Per esempio, se n è 10 e due connessioni sono stabilite con 130.192.16.201 porta 16000 e con 130.192.16.202 porta 10000, e il tempo tra l'instaurazione delle due connessioni è 25 secondi, il file di log deve contenere le linee seguenti

```
Connection accepted from 130.192.16.201 16000
No new connections for 10 seconds
No new connections for 10 seconds
Connection accepted from 130.192.16.202 10000
```

Importante: fare il flush dell'output non appena una nuova linea di output è stata prodotta.

Il/i files C del programma `server2` devono essere scritti dentro una directory `$ROOT/source/server2`, ad eccezione dei file comuni che, come già detto, devono essere messi nella directory `$ROOT/source`.
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.
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 -std=gnu99 -o socket_server1 server1/*.c *.c -Iserver1 -lpthread -lm

gcc -std=gnu99 -o socket_client1 client1/*.c *.c -Iclient1 -lpthread -lm

gcc -std=gnu99 -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 `$ROOT/source` (per esempio è possibile usare i files dal libro dello Stevens, ma questi files devono essere inclusi da chi sviluppa la soluzione).
Tutti i files sorgenti prodotti (`client1, server1, server2,` e i files comuni) devono essere inclusi in un singolo archivio zip creato tramite il seguente comando bash (lanciato dalla directory `$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 per trasferimento file TCP iterativo)

PER L'ESONERO, questo esercizio deve essere sottomesso entro il 20 maggio 2019, ore 11:59 am (del mattino).

Sviluppare un server TCP sequenziale (in ascolto sulla porta specificata come primo parametro sulla riga di comando come numero decimale) che, dopo aver stabilito una connessione con un client, accetti richieste di trasferimento di file dal client e spedisca i files richiesti indietro al client, seguendo il protocollo specificato nel seguito. I files disponibili per essere inviati dal server sono quelli accessibili dal server nel suo file system nella sua directory di lavoro.

Sviluppare un client che possa collegarsi ad un server TCP (all'indirizzo e porta specificati come primo e secondo parametro sulla riga di comando). Dopo aver stabilito la connessione, il client richiede il trasferimento dei files il cui nome è specificato sulla linea di comando dal terzo parametro in poi, e li salva localmente nella propria directory di lavoro.

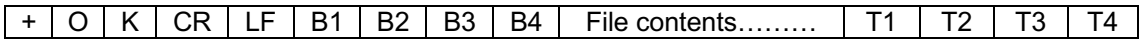
Dopo aver trasferito e salvato localmente un file, il client deve stampare su standard output un messaggio circa l'avvenuto trasferimento, includendo il nome del file, seguito dalla dimensione del file (in bytes, come numero decimale), e dal timestamp di ultima modifica (come numero decimale).

Eventuali timeout usati da client e server per evitare attese infinite devono essere impostati a 15 secondi.

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 dai caratteri ASCII carriage return (CR) e line feed (LF):

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

Il server risponde inviando:



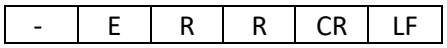
Notare che il messaggio è composto da 5 caratteri, seguiti dal numero di byte del file richiesto (un intero senza segno su 32 bit in network byte order - bytes B1 B2 B3 B4 nella figura), seguito dai bytes del contenuto richiesto, e poi dal timestamp dell'ultima modifica (Unix time, cioè numero di secondi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

Per ottenere il timestamp dell'ultima modifica al file, si faccia riferimento a

lle chiamate di sistema *stat* o *fstat*.

Il client può richiedere più file usando la stessa connessione TCP inviando più comandi GET, uno dopo l'altro. Quanto il client ha finito di spedire i comandi sulla connessione, incomincia la procedura per chiudere la connessione. In condizioni normali, la connessione dovrebbe essere chiusa in maniera ordinata, cioè, l'ultimo file richiesto dovrebbe essere stato trasferito completamente prima che la procedura di chiusura termini.

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



(6 caratteri) e quindi procede a chiudere in modo ordinato la connessione con il client.

Si usi la struttura di directories inclusa nel file zip fornito con questo testo. Dopo aver un-zippato l'archivio, si troverà una directory chiamata `lab2.3`, che include una cartella `source` con una sottocartella `server1` dove si deve scrivere il client. Sono già presenti degli scheletri vuoti di files sorgenti (uno per il client ed uno per il server). Si riempiano semplicemente questi files con i propri

programmi senza spostarli. Possono essere usate librerie di funzioni (es. quelle dello Stevens). I files sorgente C di tali librerie devono essere copiati nella cartella `source` (non metterli nelle sottocartelle `client1` o `server1`, perché tali sottocartelle devono contenere solamente il proprio codice!). Inoltre, si ricordi che se si vogliono includere alcuni di tali files nei propri sorgenti è necessario specificare il percorso "`..`" nella direttiva include).

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 –std=gnu99 -o server server1/*.c *.c –Iserver1 -lpthread -lm

gcc –std=gnu99 -o client client1/*.c *.c –Iclient1 -lpthread -lm
```

La directory `lab2.3` contiene anche una sottocartella chiamata `tools` che include alcuni strumenti di test, tra i quali è possibile trovare i files eseguibili di un client e un server di riferimento che si comportano secondo il protocollo stabilito e che possono essere usati per effettuare tests di interoperabilità.

Provare a collegare il proprio client con il server di riferimento, ed il client di riferimento con il proprio server per testare l'interoperabilità (notare che i files eseguibili sono forniti per architetture sia a 32 bit sia a 64 bit. I files con suffisso _32 sono compilati 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 tra loro al termine delle modifiche. Al termine dell'esercizio, si dovrà avere un client e un server che possono comunicare tra loro e che possono operare correttamente con il client ed il server di riferimento.

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.

Provare anche il comportamento del proprio client e server in condizioni di errore:

- Mentre una connessione è attiva provare ad attivare un secondo client verso il medesimo server. E' possibile notare che ulteriori client si collegano con il server comunque, usando il 3-way handshake del TCP anche se il server non ha ancora chiamato la funzione accept(). Questo è un comportamento standard del kernel di Linux per migliorare il tempo di risposta dei server. Comunque, fino a che il server non ha effettuato la chiamata accept() esso non può usare la connessione, ossia i comandi non sono ricevuti. In caso il server non chiami la funzione accept() per un po' di tempo, la connessione aperta dal kernel verrà automaticamente chiusa.
- Provare ad attivare sul medesimo nodo una seconda istanza del server sulla medesima porta.
- Provare a collegare il client ad un indirizzo non raggiungibile.
- 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.

Quando il test del proprio client e server è terminato, si possono usare gli script di test forniti nella directory `lab2.3` per effettuare un test finale che verifichi che il proprio client e server siano conformi ai requisiti essenziali dell'esercizio, e che passino i tests obbligatori per la sottomissione. Per lanciare lo script di test, semplicemente dare il seguente comando dalla directory `lab2.3`:

```
./test.sh
```

Lo script dirà se la soluzione è accettabile. Se non lo è, correggere gli errori e riprovare fino a quando si passano i tests. Gli stessi tests verranno eseguiti sul nostro server quando la soluzione viene sottomessa. Altre caratteristiche della soluzione sottomessa verranno testate dopo la chiusura delle sottomissioni, al fine di decidere l'esito dell'esonero ed assegnare il corrispondente voto.

Se si desidera far girare lo script di test su un sistema a 32 bit è necessario per prima cosa sovrascrivere i files eseguibili sotto la directory `tools` con la loro versione corrispondente a 32 bit. Per esempio:

```
mv server_tcp_2.3_32 server_tcp_2.3
```

Si avrà la possibilità di sottomettere la soluzione da considerare per l'esonero **insieme alla soluzione di un altro esercizio che verrà assegnato nel lab3**. Le istruzioni di sottomissione saranno fornite con il lab3.