Programmazione Distribuita I

Test di programmazione socket, 23 giugno 2014 – Tempo: 2 ore

A partire dal server di test del laboratorio 1.1, e utilizzando le soluzioni già sviluppate per l'esercizio di laboratorio 2.4, scrivere un client ed un server che si parlino tramite un protocollo di tipo richiesta-risposta specificato come segue:

a) Il protocollo si appoggia su TCP
b) Dopo il setup della connessione, il client spedisce un messaggio di richiesta, con un formato specificato dal seguente tipo XDR:

```
struct Request {
    float data<>;       /* the request data */
    operation op;       /* the requested operation */
};
```

dove `operation` è definito da

```
enum operation {
            ENCODE  = 0,
            DECODE = 1
};
```

c) Dopo aver ricevuto un messaggio di richiesta, il server invia un messaggio di risposta, con un formato specificato dal seguente tipo XDR:

```
struct Response {
    bool success;  /* true if the operation has been executed successfully */
    float data<>;  /* the response data, empty in case of unsuccessful
                      operation */
};
```

dove,

- se il formato del messaggio ricevuto è corretto e l'operazione richiesta è ENCODE, la risposta include success=true e la codifica dei dati inviati nella richiesta (la codifica è calcolata aggiungendo una costante K ad ogni numero nella sequenza)
- se il formato del messaggio ricevuto è corretto e l'operazione richiesta è DECODE, la risposta include success=true e la decodifica dei dati inviati nella richiesta (la decodifica è ottenuta sottraendo una costante K ad ogni numero nella sequenza)
- se il formato del messaggio ricevuto non è corretto, la risposta include success=false e un array vuoto

d) Dopo aver ricevuto la risposta, il client chiude la connessione.

Il server deve essere in grado di servire almeno fino a 5 client simultanei (ma eventualmente anche di più).

Il programma server deve ascoltare su un numero di porta specificato come primo argomento sulla linea di comando e deve usare un valore di K specificato in formato decimale come secondo argomento sulla linea di comando.
Il client deve ricevere l'indirizzo IP del server e il numero di porta sulla linea di comando rispettivamente come primo e secondo parametro, seguiti dall'operazione che il client deve richiedere (“ENCODE” o “DECODE”) e infine dal nome di un file di testo locale che contiene la sequenza dei numeri decimali (scritti secondo la sintassi accettata dal %f della funzione

scanf, uno per linea) da inviare come dati nella richiesta. Il client deve leggere questo file, spedire una richiesta al server con il contenuto specificato sulla linea di comando e, in caso di successo, scrivere i dati ricevuti in risposta su un file di testo locale di nome “output.txt”, un numero per linea. Invece, in caso di errore, il client deve scrivere sullo standard output un messaggio che inizia per “Error” e non scrivere il file di output.
Per esempio, un comando valido per avviare il server è:
socket_client 127.0.0.1 2048 ENCODE data.txt

I tipi XDR da usare per i messaggi sono già disponibili in un file di nome `types.x` (che si trova nella stessa directory dove è stato trovato questo file).

I files C del programma client devono essere salvati sotto la directory `$ROOT/client`, dove `$ROOT` è la directory di lavoro, i files C del programma server devono essere salvati nella directory `$ROOT/server` mentre altri files che sono comuni al client e al server devono essere salvato nella directory di lavoro. Tutti i files sorgente (client, server, e files comuni) devono essere inclusi in un singolo file zip creato con questo comando di shell (invocandolo nella cartella `$ROOT`):

```
zip socket.zip client/*.c client/*.h server/*.c server/*.h *.c *.h
```

Non includere i files usati per testare il protocollo e neppure i files generati da rpcgen, ma includere tutti gli altri files necessari per compilare il client ed il server (è possibile usare i files dal libro dello Stevens, ma questi files devono essere inclusi).
Al termine della prova, il file zip con la soluzione deve essere lasciato dove avete trovato questo file (nella directory exam_20140623).

Attenzione: controllare che il file zip sia stato creato correttamente aprendolo di nuovo e verificando il suo contenuto prima del termine della prova.
Nota: la soluzione sarà presa in considerazione solamente se è possibile compilare le applicazioni lanciando i seguenti comandi (dalla directory nella quale l'archivio è stato estratto):

```
rpcgen –h types.x –o types.h
rpcgen –c types.x –o types.c
gcc -o socket_client client/*.c *.c -I client -lpthread -lm
gcc -o socket_server server/*.c *.c -I server -lpthread -lm
```

Viene fornito un programma di test per verificare le funzionalità di base del vostro programma. Se il vostro programma non passa questi test è molto improbabile che possiate superare l'esame. In aggiunta, altri test verranno effettuati per valutare il vostro programma. Il programma di test fornito può essere lanciato così:

```
bash ./test.sh
```

Si prega di leggere il file “README.txt” dentro la cartella “DP_June14_Test” per le istruzioni su come lanciare i tests.