Programmazione Distribuita I

Test di programmazione di rete

Da sottomettere entro il 15 luglio 2013, 23.59

Sviluppare un'applicazione client-server basata sull'API dei socket che funzioni secondo le seguenti specifiche:

1. Il server è parametrizzato tramite una sequenza di numeri reali s(0),...,s(n-1). Il suo scopo è di ricevere da ogni client un'altra sequenza di numeri reali c(0),...,c(m-1) (m può essere differente per ciascun client), e calcolare e restituire a ciascun client il risultato della seguente operazione:

$$r = \sum_{k=0}^{m-1} c\ k\ s(k \bmod n)$$

2. Il server riceve esattamente due argomenti sulla linea di comando: la porta TCP sulla quale ascolta, seguita dal nome di un file di testo che contiene la sequenza dei numeri reali s(0),...s(n-1), un numero per linea. I numeri reali sono scritti nel file secondo la sintassi riconosciuta dal modificatore di formato "%f" della funzione scanf del C.

   La lunghezza della sequenza n può variare da 1 a 1000. Se il file contiene più di 1000 linee, le linee in eccesso devono essere ignorate, mentre se il formato del file non è corretto o il file non è presente il server deve terminare immediatamente.

3. Il client riceve esattamente tre argomenti sulla linea di comando: l'indirizzo IP o il nome del server, il numero di porta del server e il nome di un file che contiene la sequenza dei numeri reali c(0),...c(m-1), un numero per linea (il formato di questo file è lo stesso di quello del server ma in questo caso non c'è un limite massimo sul numero di linee). Gli argomenti sono passati esattamente nell'ordine specificato. Se il formato del file è errato o il file non è presente, il client deve terminare immediatamente.

4. Nel momento in cui la connessione TCP è stabilita, il client può iniziare a spedire una sequenza di uno o più messaggi XDR al server. Ognuno di questi messaggi include un blocco della sequenza di input c(0),...,c(m-1), ed è descritto dal tipo XDR seguente:

```
struct Request {
    float data<>;       /* il blocco dei dati */
    bool last;          /* true se e' l'ultimo blocco */
};
```

5. Il blocco in ogni messaggio è la continuazione della sequenza, cosicché il server può ricostruire la sequenza originale semplicemente concatenando i blocchi nell'ordine nel quale essi arrivano. Il client può scegliere la dimensione dei blocchi arbitrariamente.

6. Nel momento in cui il server riceve i blocchi, esso avanza immediatamente nella procedura di calcolo. Al termine, avendo ricevuto l'ultimo blocco che contiene last=true, il server invia un messaggio di risultato al client e poi termina la connessione. Il messaggio di risultato è descritto dal seguente tipo XDR:

```
struct Response {
    bool error;         /* true se e' successo un errore */
    float result;       /* il risultato del calcolo (valido solo se error
                        e' false) */
};
```

7. Se il server non può decodificare il messaggio ricevuto, deve inviare un messaggio di risultato con error=true.

8. Il client, dopo aver inviato l'ultimo messaggio con last=true, attende il messaggio di risultato e dopo averlo ricevuto chiude corrispondentemente la connessione con il server.

9. Il client e il server devono essere sviluppati per il sistema operativo Linux.
10. Il server deve essere in grado di gestire almeno 5 client simultaneamente.
11. Per semplicità, non è richiesto di implementare alcun meccanismo di timeout, né nel server né nel client.
12. Per tutto ciò che non è specificato si può scegliere liberamente come implementarlo.

I files C del programma client devono essere salvati nella directory `client`, i files C del programma server devono essere salvati nella directory `server`. Tutti i files sorgente del client e del server devono essere inclusi in un singolo archivio zip creato con in comando bash:

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

Non includere i files usati per testare i programmi, ma includere tutti i files necessari per compilare il client e il server, inclusi i files generati da rpcgen (è possibile usare i files dal libro dello Stevens, ma essi devono essere inclusi).

Il file zip con la soluzione deve essere sottomesso online prima della scadenza indicata all'inizio del testo usando il form di sottomissione disponibile dalla rete interna del Politecnico all'indirizzo https://pad.polito.it/enginframe/pd1/pd1.xml o dall'esterno della rete del Politecnico all'indirizzo https://pad.polito.it:8080/enginframe/pd1/pd1.xml.

Attenzione: il sistema di sottomissione è *automatico.* Le sottomissioni saranno automaticamente disabilitate alla scadenza. E' vivamente sconsigliato sottomettere la propria soluzione negli ultimi minuti prima della scadenza.

Nota: le sottomissioni saranno considerate valide solamente se è possibile compilare il server tramite il seguente comando (lanciato nella directory nella quale l'archivio è stato estratto, assumendo che il file types.xdr sia nella stessa directory):

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

Il giorno precedente l'esame orale tutte le soluzioni sottomesse saranno testate e ognuno riceverà un'email con i risultati dei test che verificano la propria sottomissione.

In caso di dubbi e domande relative a questo compito, per prima cosa si controllino le pagine del forum nel sito web del corso per verificare se altri hanno già fatto la stessa domanda, altrimenti si utilizzi il forum (non l'email all'insegnante) per inviare la domanda così che la risposta sia disponibile per tutti.

Si noti che le domande sul forum devono riguardare esclusivamente richieste di chiarimenti riguardo alla specifiche fornite in questo testo. Non si devono porre domande riguardanti come risolvere questo assegnamento.