ESERCITAZIONE

Dipartimento di INFORMATICA

Esercizi basati su clientUDP e serverUDP

1) Lanciare prima il server e poi il client. Cosa si osserva? Invertire la sequenza di lancio. Cosa si osserva?
2) Modificare i sorgenti per mettere il server che **riceve** sulla porta 10000 e il client che **trasmette** dalla propria porta 30000 (ogni modifica dei sorgenti richiede una loro ricompilazione)
3) Mettere il server in ascolto sulla porta 100 e osservare cosa succede
   - Bisogna modificare anche il client? Dove?
   - Per chi usa il proprio PC con Linux o una virtual machine Linux, lanciare il server con il comando "`sudo ./serverUDP`" e osservare cosa cambia
4) Sostituire "127.0.0.1" prima con "localhost" e poi con "pippo" e osservare cosa succede
5) [Da fare solo se in Lab Delta] Accordarsi per lavorare su coppie di macchine in modo che server e client siano su macchine diverse. Come bisogna modificare i sorgenti?
6) Lanciare due volte il server usando due terminali. Cosa si osserva? Funzionano entrambi?
7) Modificare il server in maniera che soddisfi 5 richieste prima di terminare
   - E se volessi che non terminasse mai?

RISPOSTA:

N.B: durante il processo, è possibile visualizzare per bene lo scambio di pacchetti desiderato attraverso WireShark e il filtro "*ip.src === 127.0.0.1 && ip.dst === 127.0.0.1*";

1. se il server è aperto la richiesta viene soddisfatta; nel caso opposto, la richiesta client rimane in attesa e non verrà mai soddisfatta, nemmeno facendo partire il server effettivamente;
2. non avviene connessione dato che la porta è diversa;
3. bisogna modificare il client per far combaciare le porte;
4. localhost è la stessa cosa di quell'IP, pippo non funziona perchè non è una connessione;
5. bisogna mettere stessa porta e l'ip della connessione
6. il secondo server termina subito probabilmente perchè vede che è già aperto;
7. fatto, basta aggiungere un while(true) prima della ricezione UDP;

Esercizi basati su clientUDP_inc.c e serverUDP_inc.c:

1) Compilare ed eseguire il secondo esempio;
2) Modificarlo per costruire una semplice sommatrice – Il client acquisisce ripetutamente da tastiera un numero intero e lo manda al server finché l'utente digita zero – Il server accumula in una variabile "somma" i valori mandati dal client finché il client manda zero – Quando il client manda zero il server risponde al client con la somma ottenuta;

RISPOSTA:
In sostanza i file originali hanno un client che chiede un numero all'utente, lo manda al server, che come sempre va eseguito prima, che lo incrementa di uno e lo rimanda indietro.

Per modificare il file è stato sufficiente aggiungere un while response != 0, creare una somma cumulativa nel server, o continuare a chiedere input all'utente nel client.

3) impossibile da fare senza ethernet;
4) Invocare il server della sommatrice con due client diversi (tutti e tre possono anche essere sulla stessa macchina ovviamente su finestre terminali diverse). Che somma leggo da ciascun client? E' la somma che ciascun client da solo si aspetterebbe?

RISPOSTA:
Il primo client che termina riceve la somma cumulativa ottenuta da entrambi i client, mentre il secondo rimane in attesa infinita.

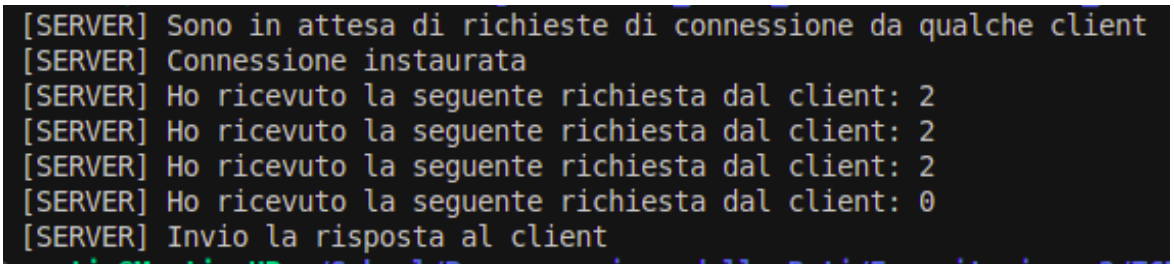
Esercizi basati su clientTCP.c e serverTCP.c

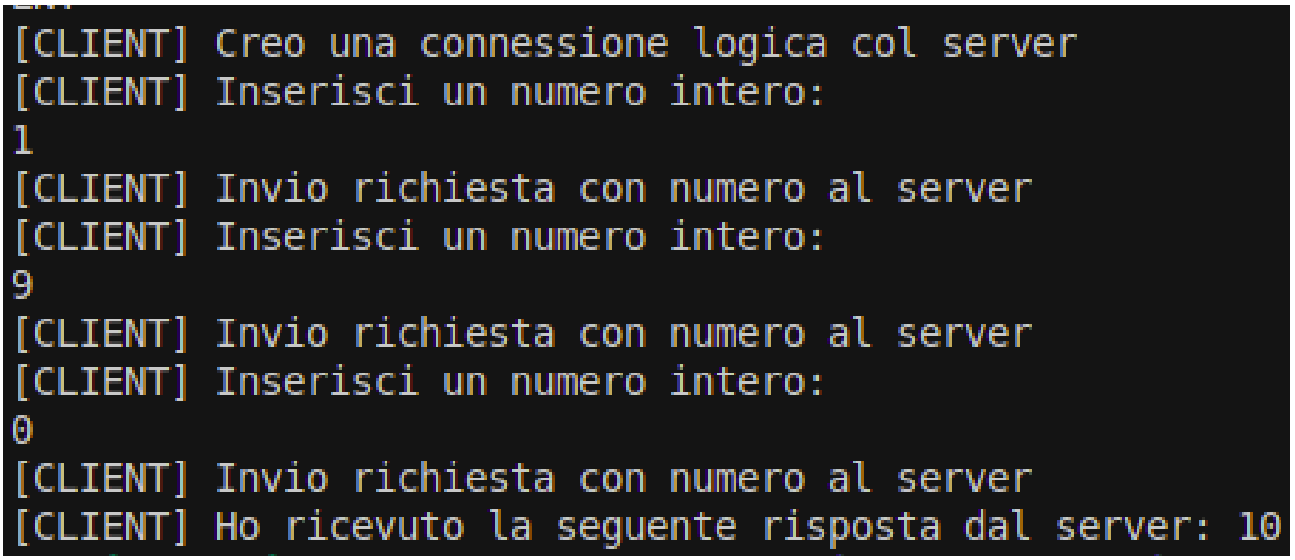
1) Lanciare due volte il server usando due terminali. Cosa si osserva? Funzionano entrambi?
2) Scrivere la sommatrice usando TCP, compilare ed eseguire
3) Provare a rifare il caso dell'Esercizio 10 ma con questa nuova versione della sommatoria. Cosa si può osservare? Che soluzione si può trovare? C'è influenza reciproca tra i due client?

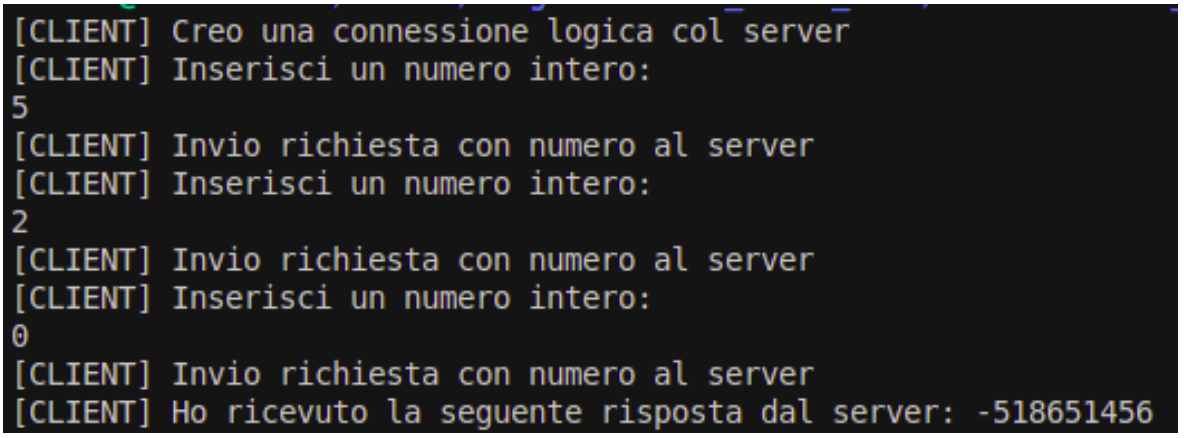
Risposta 1:
Il secondo server lanciato si chiude subito probabilmente perché la funzione di creazione del socket per prima cosa fa un controllo per la porta richiesta, e se già aperta killa il processo;

Risposta 2:
Fatto! chiaramente c'era da separare fuori dal ciclo while le richieste di apertura, la sua accettazione, le chiusure TCP e l'invio della risposta dal server al client.

Risposta 3:
Il primo client che si collega riceve effettivamente la somma cumulativa corretta, relativa ai soli dati inviati dallo stesso. Il secondo client invece (verificabile facilmente dagli output del server), invia i dati ma il server non ne salva la somma (ossia proprio non lo "caga"); l'unica cosa che funziona effettivamente è l'apertura della connessione, quindi lo scheletro iniziale del processo (già la chiusura non funziona correttamente). Come si vede dalle img, la risposta al secondo client in realtà è un errore di calcolo interno al client stesso probabilmente.







Esercizi basati su clientTCPChar.c e serverTCPChar.c

Per questi file non ci sono esercizi da svolgere. In ogni caso, il client invia un char ‘A’ e il server risponde con un char ‘B’. Estremamente statico.

Esercizi basati su clientTCPIO.c e serverTCPIO.c

1. Il client richiede al server un file specificandone il nome. Il server lo trasmette un byte alla volta. Il client lo salva in locale con lo stesso nome. Quale protocollo usiamo?

Quando devi trasferire un file come in questo caso, specialmente se è piccolo, hai bisogno che i dati arrivino in ordine, completi, senza duplicazioni, e che l'intera comunicazione sia affidabile. Ho quindi creato un server e un client che sfruttano le funzioni TCPSender e TCPReceiver della libreria network.h/c, nonchè le sys call open/read/write per la gestione del file, per far si che: client richieda nome file all’utente, lo mandi al server che attende il nome, carica il file e lo legge, mandandolo indietro al client un byte alla volta che a sua volta lo stampa su terminale.