Programmazione distribuita I

A.A. 2014-2015, Esercitazione di laboratorio n. 2

Traccia di soluzione

Esercizio 2.1 (client UDP perseverante)

Utilizzare lo stesso approccio dell'esercizio precedente, inserendo i tentativi di spedizione in un ciclo opportuno.

Esercizio 2.2 (server UDP limitante)

Utilizzare lo stesso approccio dell'esercizio sul server UDP, memorizzando ogni volta l'indirizzo di provenienza del pacchetto in un vettore di 10 elementi, che simuli un buffer circolare. Per memorizzare l'indirizzo, utilizzare un vettore di strutture sockaddr oppure sockaddr_storage cosi' da facilitare la portabilita' del codice su IPv6 (che puo' essere realizzata come utile esercizio).

Esercizio 2.3 (server TCP iterativo)

Usare le funzioni Socket(), Bind() e Listen(). Inserire la funzione Accept() in un ciclo infinito, per gestire il servizio al client successivo una volta che il primo client abbia chiuso la connessione. Dopo la Accept(), un secondo ciclo infinito riceve comandi dal client e invia il file richiesto o un messaggio di errore. Dal ciclo infinito di ricezione dei comandi si esce quando si riceve il comando QUIT (la connessione deve essere chiusa lato server tramite Close()) oppure quando non e' piu' possibile usare il socket. In quest'ultimo caso stampare il messaggio che la connessione e' stata chiusa dal client, e chiudere tramite Close() lato server la connessione. Attenzione a non utilizzare la Readline() per leggere i comandi dal client, ma realizzare in proprio il ciclo di lettura carattere per carattere fino al '\n', in quanto la Readline() bufferizza l'input e potrebbe creare problemi nelle successive Read().

Esercizio 2.4 (dati in standard XDR)

Procedere come nel caso della realizzazione di un client TCP. Creare un buffer XDR tramite la xdr_create(), inserire i numeri tramite xdr_int(), e rilevare la quantita' di byte da inviare tramite la xdr_getpos(). In ricezione, poiche' non e' nota a priori, in generale, la quantita' di byte da leggere, per mantenere il codice portabile utilizzare fdopen() per ricavare un puntatore FILE * al socket, e passarlo alla xdrstdio_create() per inizializzare la struttura dati XDR. Procedere poi con chiamate xdr_int() secondo le necessita'. Per gestire l'eventuale messaggio di errore modificare sia client sia server introducendo un'informazione addizionale che evidenzi se e' presente un errore, prima dell'intero o della stringa. Ricordarsi di rilasciare le risorse (es. con fclose()) al termine.