Esercitazione 5

Obiettivi
Utilizzo dei socket
Realizzazione di un semplice server
Gestione delle risorse

Problema

Realizzare sistema di chat di gruppo composto server e un client.

Server

1. Il server gestisce una unica chat di gruppo comune a tutti gli utenti
2. Rimane in attesa di connessioni su una porta scelta all'avvio
3. Accetta senza autenticazione la connessione di ogni client tramite un socket, che si presenta inviando il proprio nickname su un linea di testo
4. Mantiene una lista dei nickname di tutti i client connessi e degli ultimi N messaggi
5. Quando un client si connette, gli manda la lista di tutti gli altri nickname connessi
6. Quando il sistema riceve un messaggio da un client, lo invia a tutti gli altri client connessi aggiungendo il nickname di chi ha inviato
7. Permette l'invio di messaggi privati utilizzando il comando "/private" nel messaggio inviato da un client:
    a. /private nickname testo_messaggio
8. Dopo 60 secondi di inattività disconnette un client
9. discutere le possibili strategie di gestione delle connessioni dei client, tenendo conto che potenzialmente si possono avere anche molte connessioni aperte contemporaneamente e implementarne una

Client

1. I client hanno un'interfaccia basata su linea di comando, stampano i messaggi ricevuti sullo standard output e leggono dallo standard input
2. All'avvio il client riceve come parametri:
    - Indirizzo IP del server
    - Porta su cui è in ascolto
    - Nickname dell'utente
3. Fino a quando non è connesso e ha registrato con successo il nickname, il client visualizza la scritta "connecting to nome server…"
4. Terminato l'handshake iniziale, il client visualizza:
    - la lista degli utenti connessi (su una singola linea, separati da spazio)
    - gli ultimi n messaggi, uno per riga
    - da ultimo, il prompt per inviare un messaggio (quando l'utente scrive /quit esce)

5. Un programma interattivo che deve contemporaneamente leggere da console i messaggi da inviare e stampare quelli ricevuti ha il problema di evitare di mischiare input e output; ad esempio non deve stampare i messaggi ricevuti mentre l'utente sta scrivendo.
   Una tecnica per gestire queste situazioni è non appoggiarsi direttamente alla console, che ha solo un flusso lineare in "append", ma memorizzare tutti gli eventi di input/output su dei buffer interni e poi fare il refresh della console, cancellando e ristampando tutto, ogni qualvolta cambia qualcosa.
   Sotto windows si può utilizzare **conio** mentre sotto linux **ncurses**.
   https://www.programmingsimplified.com/c/conio.h
   https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/

   Con queste librerie è possibile un accesso avanzato alla console, in particolare ci interessano:
   - una funzione per leggere un carattere per volta senza attendere un newline e senza stamparlo (**getch()** funziona sia con ncurses che conio, con ncurses occorre chiamare anche noecho() all'inizio del programma oltre a initscr())
   - una funzione per cancellare lo screen (ncurses: **clear()**, conio: **clrsrc()**)
   - una funzione per scrivere una linea sullo schermo (ncurses: **addstr(char*)**, conio: **cputs(char*)**)

   Occorrono pertanto due thread:
   - uno per leggere dallo standard input ogni carattere scritto dall'utente e memorizzarlo in un buffer interno
   - uno per leggere i messaggi in arrivo dal server e memorizzarli in una apposita struttura

   Inoltre ad ogni evento di IO occorre:
   - cancellare lo schermo
   - stampare tutti i messaggi ricevuti + il messaggio che l'utente sta editando, così da rendere l'esperienza utente accettabile