Laboratorio
Programmazione concorrente e distribuita

Simone Albertini
6 giugno 2013

Come si lavora oggi

- Ognuno sulla sua postazione
- Svolgere l'esercizio sulla traccia
- Si possono scambiare opinioni, consultare materiale, fare domande.
- Tempo: 3 ore.

http://bit.ly/1aZkOdB

Problema

- *Realizzare una **chat room***
- *Un server gestisce la chat*
- *I client si connettono*
  - *Forniscono il proprio username (autenticazione)*
  - *Se ammessi, possono scrivere.*
  - *Ogni client legge ciò che scrivono tutti gli altri client connessi*

Si procede per passi

1) Implementare un server e un client base
   - Es: un server che stampa a console ciò che scrivono i client da tastiera
   - Potete prendere ispirazione dalle slide della lezione 6
   - Tre classi:
     - Client (main)
     - Server (main)
       - ServeClient: oggetto che gestisce la comunicazione con uno specifico client connesso

Si procede per passi

2) Autenticazione e tenere traccia degli utenti

- Appena un client si connette deve inviare il suo username
- Un client è ammesso se non esistono già altri utenti connessi con lo stesso username
  - E ovviamente lo username è valido (non null o empty)
- Usare un oggetto apposito che tenga traccia degli utenti connessi in un dato istante a cui si chiede se uno username è valido.

Serve un protocollo!

- Decidiamo che sugli stream inviamo solo oggetti di tipo String
- Protocollo:
  - Client: invia username
  - Server: risponde con
    - “OK” se accettato
    - Una stringa con la descrizione dell'errore se rifiutato
  - Durante la comunciazione, il client invia i messaggi letto da standard input

Cosa bisogna avere a questo punto

- A questo punto si deve avere un sistema per cui:
  - Ciascun client si connette al server di chat fornendo uno username
  - Qualunque cosa scritta da ciascun client è scritta su standard output dal server:

```
** Incoming request: simone - Connection Accepted: Socket[addr=/127.0.0.1,port=45573,localport=6666]
simone> ciao, c'e' qualcuno?
** Incoming request: gianni - Connection Accepted: Socket[addr=/127.0.0.1,port=45574,localport=6666]
gianni> sono ottimista!
simone> io no
gianni disconnected
simone disconnected
```

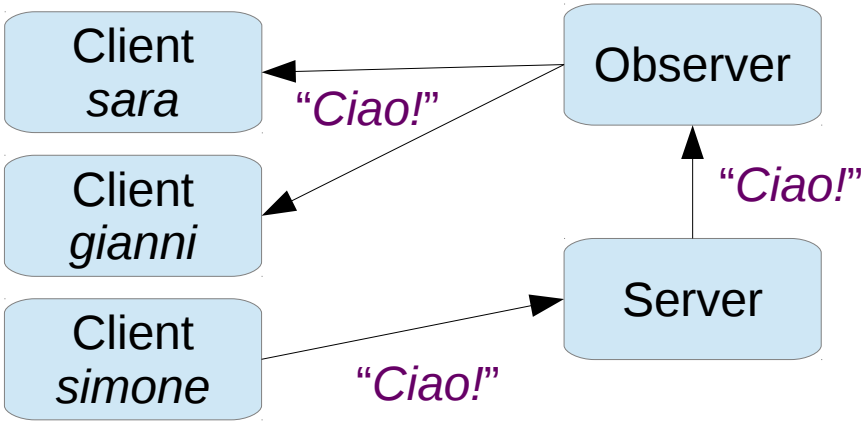
Si procede per passi

3) … ma i client non vedono cosa scrivono gli altri, che razza di chat è?

- Non è banalissimo realizzarlo.
- Bisogna sfruttare l'oggetto creato per registrare i client connessi (lato server).
- Tale oggetto deve dare la possibilità di mandare in broadcast un messaggio a tutti gli altri utenti.
- I client devono avere un Thread che rimane in ascolto e, quando arriva un messaggio dal server, lo scrivono su standard output.

Echo dei messaggi

- Il candidato migliroe è l'observer
  - Quando i client sono autenticati, vanno anche registrati presso l'observer
  - Quando il server riceve un messaggio da un client, deve inviarlo a tutti gli altri client registrati



Il server dice all'observer:
Ho ricevuto “Ciao!” da simone
Invialo a tutti gli altri client

Appendice

- Trucco per interrompere l'esecuzione di un thread.
  - Per farlo propriamente, bisognerebbe gestire le interrupt:
    - Settare un flag del thread
    - Inviare l'apposita interrupt per dire al thread di uccidersi
  - Trucco: il contenuto del metodo run ha una guardia del tipo

    “while(!stop) { <corpo run()> }”
  - E un metodo “stopMe()” che mette stop = true

Appendice

- A cosa può servirvi?
- I Client per rimanere in ascolto sull'observer potrebbero, una volta acceduti alla chat, avviare un thread il cui unico compito è di rimanere in ascolto sulla socket e ogni volta che arriva un messaggio lo scrive.
- Quando il client termina però questo thread andrebbe ucciso