UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

RETI DI CALCOLATORI
2019/2020

Documentazione

Marco Morganti

6 Aprile 2020

Indice

1 Server

1.1 Scelte Effettuate

- Multiplexing Canali mediante NIO
- Threadpool con numero fisso di Threads: 1 Thread gestisce una richiesta
- Users Online salvati in HashMap
- Users In Sfida salvati in Vector
- Comunicazione Client-Server con scrittura/lettura su SucketChannel di ByteBuffer di String
- Librerie esterne: json.simple-1.1.1.jar
- Dizionario parole esterno: dizionario_1000_parole_italiane_comuni.txt
- Per ogni Sfida: nuovo SocketChannel con relativo Selector
- Funzione hash per calcolo porta SocketChannel Sfida a partire dal nickname del mittente della richiesta
- Creazione RMI registry da codice
- N parole Diz. = 1000, K parole scelte per Sfida = 8, T. risposta Sfida = 30s, T. durata Sfida = 40s
- Pt. risposta corretta = 2, Pt. risposta sbagliata = -1, Pt. vincitore sfida = 3

1.2 Architettura Complessiva

- **Control Flow**:
  1. **Istanziamento Oggetti**
  2. **Apertura interfaccia RMI e creazione registry**
  3. **Istanziamento e Lancio Threadpool**
  4. **Lancio Server in ascolto**
- **NIO e Threadpool con numero fisso di Threads**:
  Sistema di ascolto con Clients collegati a Canali connessi al Selettore.
  Il Selettore legge le richieste dai canali e per ogni richiesta avvia uno dei thread del threadpool che si occuperà della sua gestione.
- **HashMap(SocketChannel, String) usersOnline**:
  HashMap con nickname degli utenti online e relativo SocketChannel.
  Il salvataggio dei nickname serve a garantire che 'login' sia sempre (ad eccezione di "registra_utente") la prima operazione.
  Il salvataggio del SocketChannel serve a poter gestire l'interruzione forzata del Client.
- **Vector(String) usersInSfida**:
  Vector dei nickname degli utenti già impegnati in una sfida.
  Se il Server riceve una richiesta di Sfida ed il destinatario è presente nel Vector, il Server risponde direttamente al Mittente con esito negativo senza inoltrare la richiesta.
- **Registra_utente: createRegistry(port)**:
  Così da evitare di la creazione dell'RMI registry da shell.

- **Architettura Sfida**:
  Per ogni sfida viene istanziato un nuovo SocketChannel e Selector al quale si collegano i relativi Client.
  La porta del SocketChannel è calcolata con una funzione hash a partire dal nickname del mittente richiesta.

1.3 Thread Attivati e Concorrenza

- **WorkerThread**:
  - **Ciclo di Vita**:
    1. Controllo Operazione Richiesta
    2. Gestione Operazione
    3. Scrittura esito operazione su SocketChannel
    4. Se Operazione == Sfida && Esito Positivo:
       (a) Parse di K parole a caso
       (b) HTTP.GET request di ognuna delle K parole
       (c) Creazione nuovo ServerSocketChannel e Selector Sfida
       (d) Avvia Server Sfida
       (e) Termina Sfida, Chiude ServerSocketChannel e Selector Sfida
  - **Gestione Concorrenza**:
    * **ReentrantLock lockUsersOnline**: lock/unlock per modifiche a HashMap utenti online.
    * **ReentrantLock lockUsersInSfida**: lock/unlock per modifiche a Vector utenti in sfida.
    * **Synchronized effectiveWrite**: Così da avere scritture sequenziali in database.json

1.4 Problemi Riscontrati e Soluzioni Scelte:

- **Inoltro Richiesta Sfida a User già impegnato**:
  Nel caso in cui il destinatario di una richiesta di Sfida fosse stato:

  a) In attesa di ricevere risposta a richiesta di Sfida da un altro client
  b) In attesa di input da terminale per rispondere a richiesta di Sfida
  c) Già impegnato in una Sfida

  Allora la richiesta non viene neanche inoltrata al destinatario dal Server e il mittente riceve esito negativo.
  Il Server è al corrente se il destinatario appartiene ai gruppi a), b), c) in quanto salva gli utenti nel Vector usersInSfida non appena un utente invia/riceve una richiesta di Sfida.

1.5 Indicazioni Precise Esecuzione

Eseguire prima di esecuzione Clients.
Nessun parametro da input necessario.

2 Client

2.1 Scelte Effettuate

- Interfaccia CLI
- Attesa richiesta UDP da Server tramite Thread
- Unica funzione di gestione Sfida lanciata sia da mittente (Main Process), che da destinatario (Thread)
- Utilizzo classe Timer per gestione Timeout risposta Sfida
- Utilizzo classe Future per gestione Timeout Sfida

2.2 Architettura Complessiva

- **Control Flow**:
  1. **Stampa i Comandi CLI**
  2. **Inizializza il TCP SocketChannel**
  3. **Attende input da tastiera**
  4. **Invia input al Server e riceve la Response**:
     - **Se esito Negativo**: Ne Stampa l'errore
     - **Se esito Positivo**: Stampa il messaggio di risposta
     - **Se operazione era 'login'**: Lancia ClientUDPThread

2.3 Thread Attivati e Concorrenza

- **ClientUDPThread**:
  1. **Attende la richiesta UDP dal Server**
  2. **Legge Input da tastiera (Risposta: si/no)**
  3. **Invia Risposta UDP al Server**
  4. **Se Risposta == si: Inizia Sfida**

2.4 Problemi Riscontrati e Soluzioni Scelte:

- **Gestione Timeout risposta Sfida**:
  La scelta di questa soluzione inizia dal Server, che non inoltra la richiesta di Sfida al destinatario della richiesta se questo è in attesa di iniziare/ha iniziato una Sfida con un altro utente.
  Il motivo di questa scelta è che non si avrebbe sincronizzazione dei tempi di risposta alla richiesta di Sfida.
  Lato Client viene attivato un Timer che passato il tempo di attesa risposta cambia il valore di una variabile booleana, così anche se il Thread è bloccato in attesa di input da terminale per l'intera durata del tempo Timeout, una volta uscito non effettua alcuna operazione indipendentemente dall'input dato dall'utente e si rimette in attesa di richieste UDP.

2.5 Indicazioni Precise Esecuzione

Eseguire dopo esecuzione Server.
Nessun parametro da input necessario.

3 Descrizione Sintetica Classi Definite

- **Request**: Contenente richiesta operazione. Da Client a Server
- **Response**: Contenente esito di Request. Da Server a Client
- **SfidaCommunication**: Contenente parole da tradurre e traduzioni. Bidirezionale Client-Server
- **SfidaOutcome**: Contenente punteggi finali sfida. Da Server a Clients
- **TimeLimitedCodeBlock**: Di supporto per gestione tempo massimo a disposizione per Sfida
- **Operations**: Contenente metodi per gestione richieste
- **JSONWindow**: Contenente metodi per scrittura/lettura di database.json
- **Registra_utente_Remote**: Contenente metodo gestione operazione RMI registra_utente

4 Database

- **1 unico file .json**
- **Informazioni persistite**:
  - **Nickname**
  - **Password**
  - **Score**
  - **Friends**

5 Come Compilare ed Eseguire Progetto

- **Importa progetto IDE su eclipse**:
  1. **"File"**
  2. **"Open Projects from File System"**
  3. **"Directory"**
  4. **Select the directory "Word Quizzle"**
  5. **"Finish"**
- **Non rimuovere .jar file**
- **Non rimuovere dizionario .txt**
- **Lancia prima Server e poi Clients**