Progetto Reti 2019/2020
Word_Quizzle

Giorgio Pirina Matr 550167

Introduzione

Il progetto consiste in un applicazione client-server di nome Word_Quizzle in cui più clients interagiscono con un server NIO e si sfidano in una gara di traduzione italiano-inglese. Ho distinto all'interno del progetto 4 package,uno per il client,uno per il Word_Quizzleserver , un package constantRMI in cui si trovano le costanti utilizzate per il progetto e l'interfaccia comune sia al client che al server per il servizio di registrazione mediante RMI ed infine un package chiamato userTools in cui si trovano le strutture dati con cui il server gestisce I clients.Vi è un file italianWords.txt in cui si trovano circa 900 parole italiane,che saranno quelle sottoposte nelle sfide e un file JSON, Database.json

Come libreria JSON viene utilizzata Jackson 2.11.1 la libreria JSON più diffusa nonché anche attualmente l'ultima versione.

L'interfaccia è a linea di comando,poichè mi sembrava la più coerente per un gioco di traduzione in cui lo scopo sia quello di imparare a tradurre da italiano a inglese,in cui quindi la parte grafica gioca un ruolo di importanza minore.

Comunicazione Client-server

Il client invia una richiesta al server consistente in una stringa codificata in UTF-8 in cui la prima parte è un header di un byte e la seconda(il corpo) è una stringa a lunghezza variabile.Le possibili richieste sono:

- LOGIN username password, esegue il login.
- CHALLENGE username, sfida un amico a una partita.
- ACCEPTMATCH username, accetta la sfida inviata da username.
- GUESSWORD, comunica al server la traduzione inserita dall'utente
- TIMEOUT, comunica al server che la partita dell'utente è finita per time out.
- ADDFRIEND, aggiunge friend agli amici.
- SHOWFRIENDS, richiede una stringa JSON contenente I propri amici.

- SHOWRANKING, richiede una stringa JSON contenente la classifica di se stessi e i propri amici.
- SHOWSCORE, richiede una stringa JSON contenente il proprio punteggio.

  Per la de/serializzazione degli oggetti JSON viene utilizzata la classe ObjectMapper appartente alla libreria jackson.

Mentre le ultime 4 richieste vengono facilmente soddisfatte dal server inviando un messaggio di risposta(Response) in cui il corpo é una Stringa o Array in formato JSON che viene poi deserializzato e printato sullo stdout,le prime quattro vengono esaminate in dettaglio poichè richiedono più passaggi non banali.

- LOGIN:La fase di login viene implementata come da specifiche mediante RMI. Nel package CostantRMI vi è l'interfaccia Registration Service che chiaramente estende Remote. La funzione utilizzata,registerUser, ritorna un booleano che restituisce vero se la registrazione è avvenuta con successo,falso altrimenti.
- CHALLENGE:Il client digita il comando sfida Nome utente lanciando una sfida all'amico(se non è amico il server comunica al client che non sono amici) e il server comunica all'amico sulla porta UDP come da specifiche la richiesta di sfida da parte dell'utente. Qualora il client fosse già occupato in una sfida il server comunica allo sfidante l'impossibilità della sfida. La richiesta di sfida è bloccante. Lo sfidante si mette infatti in attesa bloccante in attesa della risposta dello sfidato. Qualora lo sfidato non risponde entro 10 secondi o si disconnette il server invia un messaggio allo sfidante di nome user +unavailable
- ACCEPTMATCH:una volta arrivato il datagramma UDP allo sfidato viene richiesto se accettare la sfida o meno. In caso affermativo viene inviato una ACCEPTMATCH al server che provvede a creare un istanza della classe match in cui il challenger e il challenged sono I nomi utenti dei rispettivi partecipanti.
- GUESSWORD:Una volta cominciata la sfida ad entrambi I concorrenti vengono sottoposte le stesse parole ed entrambi una volta digitata la parola la inviano al server con send Request con argomento GUESS WORD, e answer ossia il tentativo di risposta.il server provvederà a inviare la prossima parola qualora il timer non fosse ancora scattato.

- TIMEOUT:dopo aver comunicato al server che la sua partita si è conclusa per timeout, qualora il suo avversario non avesse terminato la partita l'avversario aspetta in attesa bloccante che l'altro avversario termini. Questa scelta è dovuta anche ad un uso ipotetico che potrebbe essere fatta dell'applicazione. Come nelle verifiche infatti si aspetta prima che anche tutti gli altri finiscano la prova per poi sapere il risultato e soprattutto in questo caso essendo una sfida I punteggi devono essere confrontati e quindi il server necessita di entrambi I risultati per calcolare il punteggio

TUTTE I COMANDI E LE TRADUZIONI TERMINANO CON IL CARATTERE \n

SERVER NIO

La classe **WordQuizzleServer** realizza un server non bloccante con Selector,costrutto meccanismo analogo alla select() con il quale ho implementato il mio progetto di Sistemi operativi, e si mette in ascolto sulla porta 1535, sia dell'IP di loopback che di quello della rete locale, se possibile. Per mantenere lo stato dell'utente e l'avanzamento dei dati tramessi e da trasmettere, il server associa a ogni SocketChannel una User_Session, che contiene ByteBuffer per la richiesta e per la risposta, un campo "nickname" (solo dopo il login) e un campo "currentMatch" (solo durante una partita).

Alla creazione del server, esso si mette in ascolto sulla port UDP per gestire eventuali richieste di sfida, recupera il Database JSON con la funzione parseFromFile che attraverso un Object Mapper(strumento per serializzare/deserializzare oggetti JSON) converte il database Json in una istanza di User_Manger in cui vengono gestiti gli User,la classe che rappresenta un client con I campi nickname password score e friends. Il file JSON si chiama **Database.json** e contiene inizialmente 4 user,classicamente alice bob charlie e danny.

Channel_suspended

Alla richiesta di A challenge B,A viene sospeso, ovvero deregistrato dal selector, in attesa che B accetti o meno la sfida. I canali sospesi vengono conservati in una Map<String, SuspendedUser>, dove SuspendedUser contiene il canale e la UserSession che era l'attachment della key cancellata.

Il server implementa 2 metodi, **suspendChannel**(SelectionKey key,int timeout) e **restoreWaitingChannel**(String username). Il primo appunto sposta il canale

dal KeySet del selector alla HashMap dei suspendedUsers, e inoltre avvia un thread di timeout. Quando questo thread raggiunge il timeout, esso risveglia il canale sospeso (se non era ancora stato svegliato). Il metodo **restoreWaitingChannel**, similmente, risveglia il canale sospeso, se non è stato precedentemente risvegliato dal timer. La differenza consiste nella risposta che viene comunicata al client: essere risvegliati per timeout comporta un codice di errore.

La classe **Match** viene utilizzata per gestire la sfida ed è da essa che si recuperano le parole dal file italianWords.txt. Inutile infatti sarebbe occupare memoria prima che una istanza della classe match venga istanziata. Essa viene infatti istanziata solo in presenza di una sfida. Quindi si trova anche il metodo Obtain_Translation che recupera la traduzione dal servizio Mymemory attraverso URL dato nella specifica. Questa funzione chiaramente per funzionare necessita di una connesione internet.

Un accenno sulla gestione dei punteggi:

+ 3 a risposta esatta e –1 a sbagliata. Bonus di + 3 a vincitore e –10 se l'utente lascia la partita prima del tempo.

Il server Inoltre gestisce le richieste sopra elencate(LOGIN,ADDFRIEND....etc). All'interno del server vi è inoltre una semplice CLI richiesta come da specifiche dove si possono vedere gli users in generale,quello online oppure come normalità,si può far terminare il server con la chiamata exit.

Infine per la registrazione RMI il server contiene la classe Registration_Server in cui è implementata la semplice funzione public boolean registerUser(String nickname, String password) throws RemoteException che permette attraverso nickname e password di registrare l'utente desiderato.

GESTIONE DEI THREADS,CONCORRENZA E SFIDA

Il CLient ha 3 thread attivi:

- Il main thread che è in ascolto sullo standard input per ricever token che elaborerà con la funzione elabora().Il main thread gestisce la partita

attraverso la funzione playmatch(). Questa funzione condivide con I thread timer le variabili MatchOn e la variabile timeout. Utilizzata una variabile lock su entrambe le variabili per evitare una race condition.

- Un thread in attesa bloccante sulla porta UDP in attesa di richieste di partita. Per fare ciò il thread modifica la variabile newchallenge che verrà poi letta dal Main Thread per richiedere allo user se intende accettare la richiesta di sfida o meno. Per gestire la race condition su questa variabile ho utilizzato una lock esplicita(lock1) che permette ai thread di evitare che la richiesta UDP venga persa(o meglio che non venga eseguita subito,contrariamente alla mia scelta progettuale).
- Un thread Timer che implementa l'interfaccia Runnable che condivide con il Main thread le variabili sopracitate gestite con il costrutto synchronized(lock) esattamente come nella funzione playmatch()

Il server ha I seguenti thread attivi:

- Il Main Thread, che interagisce con i client
- Eventuali thread timer per gli utenti sospesi
- Eventuali thread per l'interrogazione di MyMemory durante un Match
- Il thread in ascolto per la CLI

Il Main Thread e I timer hanno in comune la risorsa condivisa la map dei suspended user. tutti i metodi per modificarla o accedervi, quali suspendChannel, restoreWaitingChannel, endMatch, isChannelWaiting, hasWaitingUserDisconnected,timeOutChannel sono metodi synchronized, quindi richiedono il lock sull'oggetto server stesso. Anche la sezione per elaborare la richiesta ACCEPTMATCH (e quindi risvegliare un client sospeso) è contenuta in un blocco synchronized(this). Il thread che interroga MyMemory, detto translationThread, condivide un'unica risorsa col main thread, translationList. L'accesso a tale risorsa è però sequenziale, infatti quando entrambi gli utenti hanno concluso la loro partita, il main thread, prima di leggere translationList, esegue una translationThread.join().

La CLI puó leggere soltanto e non modificare 3 risorse: lo userDataManager, il Key Set del selector, e i SuspendedUsers.Per il primo e per il terzo tutti I metodi come precedentemente fatto notare sono synchronized. Per il secondo invece la CLI necessiterebbe della lock sul keyset per dare una risposta atomica sugli user attivi.In questo caso quindi si lascia una piccola

incosistenza per evitare di dare la lock sul key set, creando quella che viene chiamata “eventual consistency” e lasciare quindi che il Main thread agisca indisturbato sulla struttura e sia quindi più efficiente pagando però in consistenza (Cap theorem studiato a Cloud and green computing) evitando così una strong consistency con prestazioni minori.

SFIDA

Uno user A invia un messaggio di CHALLENGE al server via UDP e gli comunica il nickname dello sfidante con cui intende competere(lo chiameremo B).

Sulla porta UDP di B viene mandato dal server un datagramma UDP con il nome dello sfidante A. In questo caso il thread in ascolto sulla porta UDP comunica al Main Thread,modificando la variabile newchallenge,che vi è una nuova sfida.Supponendo che B accetti,lo stesso B invia al server una richiesta di ACCEPTMATCH.Il server risveglia il canale associato alla Socket dello sfidante che era stato precedentemente sospeso.Inizia così la sfida,con entrambi I client che invocano la funzione playmatch();entrambi avviano il timer di 30 secondi(tempo a disposizione per il match) e aspettano le traduzioni dal server;il server invia contemporeanamente a ciascun giocatore la parola da tradurre .Le parole da tradurre sono 7 e vengono scelte dal server dal file italianWords.txt, dopo che sia stata creata un istanza di match., vi è circa quindi 4 secondi di tempo per tradurre a parola. Ad ogni traduzione il client invia al server il proprio tentativo di traduzione con una richiesta GUESSWORD. Il server controlla che non sia scattato il timer ed invia ai giocatori la parola successiva. Chi finisce per primo deve aspettare che l’altro giocatore abbia finito la sfida. Non appena entrambi I giocatori hanno terminato e premuto invio entrambi, il server provvede ad inviare le statistiche della partita come da specifica.Da notare che l’ultima richiesta GUESS o la richiesta TIMEOUT possono essere “bloccanti”, se l’altro utente non ha terminato.Infatti se il primo giocatore che finisce si sospende per 20 secondi fino a che l’avversario non termina

 Fine
Quando il secondo utente termina, che sia con GUESS o con TIMEOUT, allora risveglia il canale del primo utente, e calcola i risultati della partita. Per ogni parola tradotta correttamente si assegnano 3 punti, per ogni errata

-1, e al vincitore 3 punti aggiuntivi. Nel caso in cui un giocatore si disconnetta, mentre gioca oppure mentre è in attesa, l'avversario viene dichiarato vincitore a tavolino e per l'abbandono viene inflitta una penalitá

ISTRUZIONI DI COMPILAZIONE

Per il server, è necessario compilare ed eseguire org.Word_Quizzle.server.ServerMainClass.java. Per il client, è necessario compilare org.WordQuizzle.client.ClientMainClass.java, ed eseguirlo passandogli come argomento l'indirizzo IP del server. Se non viene fornito nessun argomento, il client prova a connettersi all'indirizzo di loopback. Per compilare ed eseguire gli applicativi, devono essere presenti nel classpath i 3 jar di Jackson come libreria esterna oltre chiaramente alla JDK, il file "Database.json" e "italianWords.txt", consegnati insieme a

FINE