Sommario

1 Generalità 2
2 Architettura del sistema 2
2.1 Funzionalità dei thread 3
2.2 Informazioni memorizzate nel sistema 4
2.3 Gestione della concorrenza 5
2.4 Comunicazione IPC 5
2.5 Altre scelte di progetto 6
3 Scelte implementative 6
3.1 Implementazione strutture dati 6
3.2 Implementazione per connessioni di IPC 7
3.3 Registrazione 7
3.4 Creazione di un documento 7
3.5 Edit e End-Edit e Show 8
3.6 Gestione inviti e notifiche 9
3.7 Chat 9
4 Classi principali 9
4.1 Client 9
4.2 Server 11
5 Istruzioni per l'esecuzione 13
5.1 Assunzioni sull'interfaccia Client 13
5.2 Istruzioni per la compilazione 14
5.2 Esecuzione dello script di test 14

1 Generalità

L'applicazione è stata realizzata in ambiente Windows, quindi si suggerisce di eseguire eventuali test nell'ambiente target.

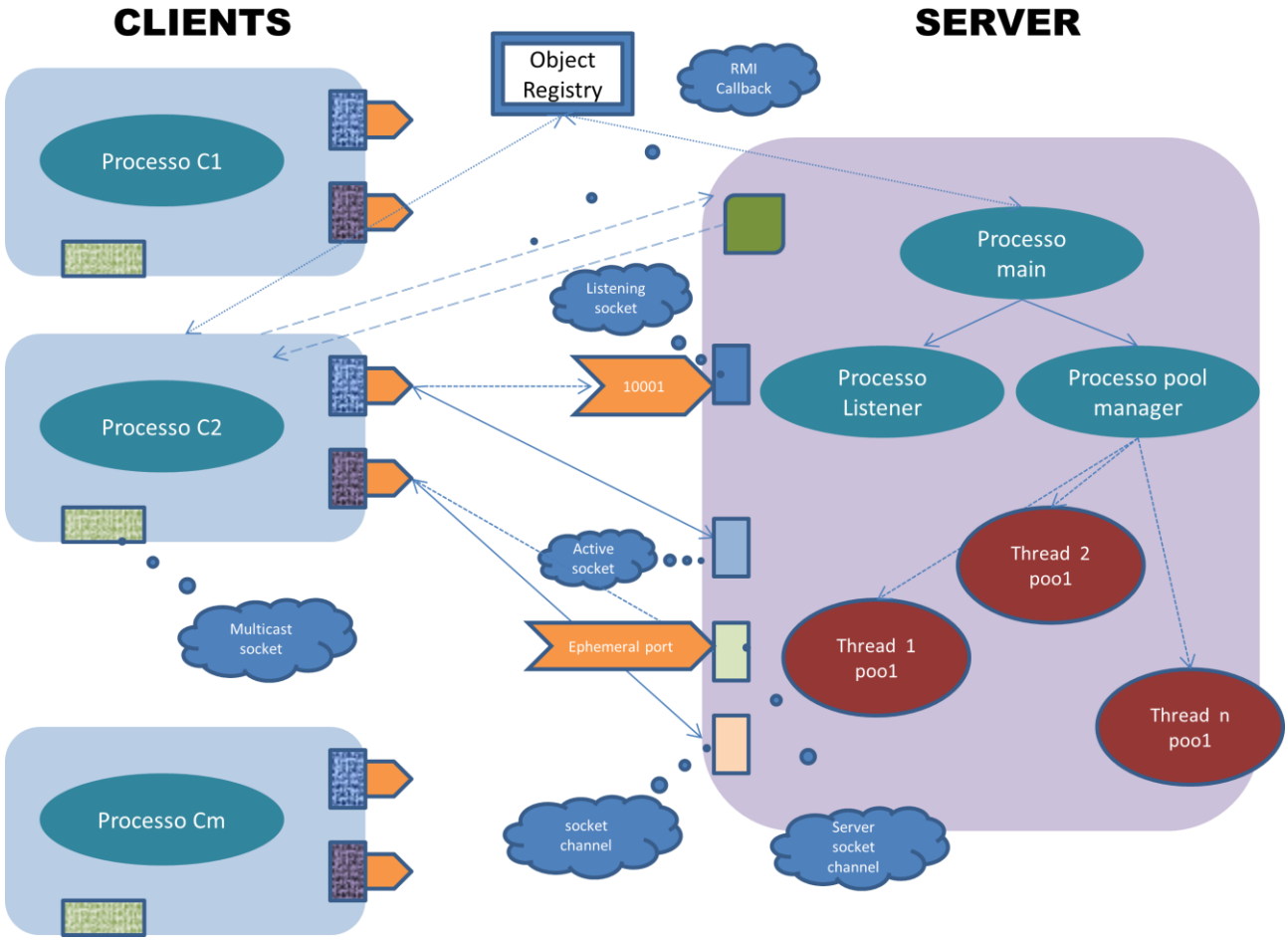
Il Client e Server Turing devono essere eseguiti sullo stesso host in quanto comunicano mediante indirizzo di loopback.

L'applicazione Client è stata sviluppata mediante un'interfaccia CLI.

2 Architettura del sistema

L'architettura utilizzata dall'applicazione Turing è Client-Server. Per massimizzare l'efficienza, Turing si serve di multithreading.

Di seguito è disponibile uno schema architetturale esemplificativo; si noti che pur essendo mostrate in figura solamente le interazioni tra un Client e il Server, gli altri Client avranno interazioni analoghe.



2.1 Funzionalità dei thread

Turing si serve di un thread listener, un threadpool e un thread cleaner:

- **thread listener**, sempre in attesa di nuove connessioni in entrata, genera socket per la comunicazione coi vari Client
- **threadpool**, una volta ottenuti tali socket, gestisce parallelamente le richieste in entrata (c'è quindi un rapporto 1:1 tra i thread del pool per la gestione delle richieste e Client in comunicazione col Server. Non sempre un Client durante la sua vita è in comunicazione col Server, ma tale eventualità verrà chiarita più avanti nella relazione alla sezione 2.4).
- **thread cleaner**, agisce periodicamente, liberando le sezioni dei documenti che sono state ottenute per l'editing da un utente per un tempo maggiore rispetto a quello consentito rimettendole in stato editabile (si ricorda che una sezione può essere editata da un solo utente alla volta; il cleaner evita che il comportamento inaspettato di un Client non renda più fruibile la sezione che stava editando).

2.2 Informazioni memorizzate nel sistema

I dati fondamentali sono relativi agli utenti, ai documenti e alle sezioni.

UTENTE

Username	PW	Flag on-line	Flag di editing	Elenco documenti creati	Elenco Documenti a cui è stato invitato	Elenco Documenti invitato ma non notificato

Alla registrazione un utente è in stato offline e non-editing, e ha tutti gli elenchi dei documenti vuoti.

Quando effettua l'operazione di login viene settato il flag on-line; tale flag viene resettato con l'operazione di logout. Il flag di editing viene settato in seguito all'operazione di Edit e resettato in seguito all'operazione di End-Edit.

DATI DEL DOCUMENTO
Nome
Username del creatore
Elenco utenti invitati
Numero utenti attivi sul documento
Numero sezioni
Elenco sezioni

DATI DELLA SEZIONE
Numero sezione
Username di chi effettua la modifica
Flag di edit
Timestamp relativa all'ultima richiesta di edit

Si noti che tali informazioni hanno valore solo a livello logico, l'implementazione verrà discussa più avanti e per avere maggiori dettagli potrà essere consultata nel codice.

2.3 Gestione della concorrenza

Ho affrontato la gestione della concorrenza sull'accesso alle strutture dati servendomi delle classi Java del package "java.util.concurrent.*"; dello specifico utilizzo si parlerà più avanti nella sezione delle scelte implementative.

Per evitare race conditions da parte dei vari thread sullo stato Online/Offline e Edit/Not Edit di un utente, sullo stato di Edit/Not Edit di una sezione e sullo stato inUse/not inUse di un indirizzo di multicast sono state utilizzate lock rientranti.

Inoltre, essendosi resa necessaria la gestione della concorrenza anche in fase di download e upload dei file corrispondenti alle varie sezioni, ho utilizzato una Read/Write lock. Si ricorda che una lock di questo tipo permette l'accesso in lettura che può essere contemporaneo da parte di più thread, l'accesso in scrittura invece è esclusivo sia rispetto ad altri scrittori che rispetto ai lettori.

2.4 Comunicazione IPC

La registrazione di un utente a Turing avviene tramite RMI, come richiesto.

La comunicazione delle richieste da Client a Server che avviene tramite socket inizia con la fase di login e termina con la fase di logout. Inoltre, il socket viene chiuso anche quando il Client passa in stato di Edit. La comunicazione viene automaticamente riaperta al termine dell'operazione di End-Edit eseguita dell'utente.

Ho fatto questa scelta perché ritengo che un utente di Turing trascorra la maggior parte del tempo ad editare una sezione di un documento, quindi l'overhead introdotto dalla riconnessione durante l'operazione di End-Edit è preferibile per il Server rispetto a mantenere attive un numero molto alto di connessioni contemporanee.

Un utente con la login si registra anche al servizio di callback (notifica all'utente di un invito ricevuto mentre è online) fornito da Turing. Ho scelto questa modalità e non una socket esplicita come richiesto perché ho ritenuto fosse la tecnologia più adatta a svolgere il compito di servizio di notifica asincrona.

Client e Server comunicano anche su una socket dedicata per il trasferimento file, costruita on-demand alla richiesta da parte del Client di editare una sezione o di visualizzarne il contenuto.

Un Client che sta editando una sezione di un documento comunica con altri Client che stanno editando altre sezioni dello stesso documento attraverso un multicast socket.

2.5 Altre scelte di progetto

Il Server Turing una volta avviato resta sempre attivo. Lo stesso vale per il Client visto che come l'automa a stati finiti suggerisce non è prevista la terminazione del programma. Si noti che durante la stessa esecuzione del Client, se non ci sono errori inaspettati, è possibile dopo il logout di un utente fare login con un altro utente.

Nella realizzazione di Turing sono state fatte stime sul numero di utenti registrati, connessi in un dato istante, numero di documenti creati. Queste stime non vogliono essere realistiche e rispondere ad una vera esigenza di mercato, perché tale compito esula dallo scopo del progetto.

Allo stesso modo sono stati tralasciati controlli di sicurezza approfonditi e altri accorgimenti fondamentali in un contesto reale di applicazione ma non funzionali al progetto in sé.

3 Scelte implementative

Di seguito sono riportate alcune importanti scelte in fase di implementazione del progetto.

3.1 Implementazione strutture dati

Per quanto riguarda le strutture dati ho preferito strutture concorrenti alle loro alternative sincronizzate. Esse sono infatti maggiormente scalabili e si adattano meglio al multithreading in quanto, quando c'è un accesso, non lockano tutta la struttura dati ma soltanto una porzione, quindi possono esserci accessi concorrenti in parti diverse della struttura contemporaneamente senza sollevare race conditions.

Per la memorizzazione degli utenti è utilizzata una ConcurrentHashMap<String, Utente> dove vengono memorizzate delle coppie <username, oggetto Utente corrispondente>.

Analogamente per i documenti viene utilizzata una ConcurrentHashmap<String, Documento> dove vengono memorizzate coppie

<nomedocumento, oggetto Documento corrispondente>. Entrambe le HashMap sono inizializzate con (numero chiavi stimate * 4/3 + 1). Questo perché si vuole evitare l'espansione della HashMap che avviene quando i 3/4 della tabella sono pieni.

Per i socket di comunicazione con i Client che il listener inserisce e i thread del pool estraggono, ho scelto di usare una ArrayBlockingQueue<Socket>.

Per la gestione degli indirizzi di multicast ho creato la classe MulticastArr; maggiori dettagli saranno dati nella sezione 3.7.

3.2 Implementazione per connessioni di IPC

Come da specifica d'implementazione l'operazione di registrazione è implementata mediante Java RMI.

Come da specifica d'implementazione le operazioni di login, richiesta e risposta di apertura/modifica di un documento, di invito sono implementate con TCP. Diversamente dalla specifica, la notifica d'invito ad un utente online è implementata con una callback.

Per quanto riguarda i file, essi sono letti tramite FileChannel e trasferiti da Server a Client richiedente (e viceversa) tramite SocketChannel. In particolare, il Server ad una richiesta legale di Edit/End-Edit/Show apre un ServerSocketChannel in ascolto, comunica sul socket TCP prima di chiuderlo la porta del ServerSocketChannel e si prepara a inviare/ricevere il file su un SocketChannel.

Ogni operazione di richiesta/risposta tra Client e Server ha il suo formato di scambio dei messaggi.

3.3 Registrazione

Lo username dell'utente è univoco e viene controllato in fase di registrazione.

3.4 Creazione di un documento

Se un Client crea un documento specificando come numero di sezioni un numero minore o uguale a zero, Turing crea il nuovo documento con 5 sezioni di default.

Nel Server il nome del documento diventa il nome di una cartella e ogni sezione corrisponde ad un file che ha come nome il numero della sezione.

Es. se il documento si chiama “prova” ed è composto da 3 sezioni, verrà creata una cartella prova che contiene il file di nome 0, 1 e 2

prova\0

prova\1

prova\2

3.5 Edit e End-Edit e Show

La End-Edit rende permanente la modifica effettuata dal Client su una particolare sezione. A tal fine il Server rimuove il file corrispondente alla sezione e crea un nuovo file con lo stesso numero all’interno della stessa cartella e con il contenuto aggiornato dal Client.

In caso di chiusura del Client mentre ci sono file in Edit sarà l’utente a dover ripulire la cartella dei download dai file rimasti. Il Server, invece, implementa un sistema di cleaning : un thread in un ciclo sempre attivo, dopo aver dormito per SLEEPING_TIME si sveglia e iterando sui documenti cerca se c’è una sezione che è in Edit da più tempo di MAX_EDIT_TIME. In tal caso la sezione viene liberata, e l’utente che la stava editando viene messo in stato di offline. Ogni tentativo successivo di End-Edit da parte di questo utente verrà ignorato.

La Show di una sezione crea in locale un solo file, mentre la Show dell’intero documento crea un insieme di file ciascuno con il nome del documento seguito dalla sezione corrispondente. Il file creato è seguito da un time stamp, in modo tale che successive Show non debbano richiedere l’eliminazione di versioni precedenti.

Es. si supponga che il documento prova sia composto da 3 sezioni

turing show prova 1 //crea 1 file

prova1 2019.01.02.10.12

turing show prova -1 //crea 3 file

prova0 2019.01.02.10.15

prova1 2019.01.02.10.15

prova2 2019.01.02.10.15

(Per dettagli sul funzionamento della Show consultare la sezione 5.1)

3.6 Gestione inviti e notifiche

In fase di login un Client riceve l'elenco dei documenti a cui è stato invitato alla modifica mentre non era loggato.

Quando un Client è online e sta compiendo delle operazioni, come già spiegato l'invito gli è notificato tramite RMI callback.

L'informazione relativa agli utenti che possono modificare un documento è replicata per rendere più veloce i controlli in fase di esecuzione. Infatti ogni qualvolta un utente viene invitato a modificare un documento, il Server modifica:

1. Nel documento, la lista degli utenti che possono modificarlo
2. Nell'utente invitato, l'elenco dei documenti che egli può modificare. Inoltre, se l'utente al momento dell'invito era offline, il nome del documento viene anche aggiunto all'elenco dei documenti al quale è stato invitato ma del quale non ha ancora ricevuto notifica.

3.7 Chat

Per gli indirizzi di multicast ai quali si appoggia la chat sono stati scelti staticamente 256 indirizzi (da 239.0.0.0 a 239.0.0.255). Essi sono nel blocco degli indirizzi destinati all'uso locale e non assegnati ad altri servizi.

Un indirizzo di multicast viene assegnato ad un documento e marcato in MulticastArr come utilizzato quando c'è almeno un utente ad editare una sua sezione. Quando non c'è più nessun utente ad editare sezioni di un dato documento, tale indirizzo viene marcato come libero.

4 Classi principali

4.1 Client

MainClassTuringClient

Classe principale del Client che viene eseguita per prima, compie un while(true); è in attesa sull'input dell'utente. Si occupa di inviare le varie richieste al Server.

ConnectionsClientSide

Si occupa di ricevere e gestire ed eventualmente stampare a video le risposte dal Server per le notifiche ricevute mentre un utente era offline, per la ricezione della lista (documenti creati e documenti ai quali l'utente è stato invitato), per la ricezione e l'invio del file corrispondente ad una sezione e per ricevere le informazioni per aggiungersi alla chat del documento.

NotifyInviteInterface

Interfaccia per l'operazione remota di notifica d'invito mentre l'utente è online.

NotifyInviteImpl

Implementazione dell'interfaccia precedente.

OpCodes

Enum che contiene i codici di operazione che il Server deve gestire e quelli di risposta. E' presente una classe analoga nel Server.

Registration

Interfaccia del servizio RMI di registrazione del Server di cui il Client ha bisogno in fase di compilazione.

4.2 Server

MainClassTuringServer

Classe principale del Server che viene eseguita per prima, inizializza le strutture dati per la memorizzazione degli utenti, dei documenti, dei socket e degli indirizzi di multicast. In questa classe viene creato ed esportato il registry per RMI. Vengono anche inizializzati tutti i thread necessari al corretto funzionamento del programma.

Clean

Task eseguito dal thread cleaner; la sezione 3.5 spiega come agisce.

ConnectionsServerSide

Si occupa di gestire la ricezione delle informazioni da parte del Server per le richieste del Client di: login, create, share, edit, end-edit, list, show sezione, show documento.

Documento

Vi sono memorizzate tutte le informazioni relative ad un documento. Tra i vari metodi ci sono quelli per ottenere tali informazioni.

Execution

Task eseguito dai thread del pool, è in un ciclo e si occupa di verificare se le richieste dei Client possano essere soddisfatte e scrivere in caso affermativo o negativo sulla socket relativa a tale Client il codice di risposta presente in OpCodes.

Listen

Task eseguito dal thread listener, è sempre in attesa di nuove connessioni da parte di Client. Quando gli arriva una nuova connessione inserisce il socket associato su una coda.

MulticastArr

Si occupa di gestire gli indirizzi di multicast. Ha metodi per controllare se un indirizzo è in uso, per assegnare un indirizzo libero e per liberare un indirizzo occupato.

NotifyInviteInterface

Interfaccia per l'operazione remota di notifica d'invito mentre l'utente è online. Il Server ha bisogno di tale interfaccia in fase di compilazione.

OpCodes

Enum che contiene i codici di operazione che il Server deve gestire e quelli di risposta. E' presente una classe analoga nel Client.

PoolManager

Task eseguito da un thread nel main, crea il thread pool e in un ciclo estrae un socket dalla coda e fornisce al pool un nuovo task Execution da eseguire.

Registration

Interfaccia del servizio RMI di registrazione offerto dal Server; contiene la registrazione a Turing, la registrazione al servizio di callback e la deregistrazione da quest'ultimo.

RegistrationImpl

Implementazione dell'interfaccia precedente

Sezione

Vi sono memorizzate tutte le informazioni relative ad una sezione. Tra i vari metodi ci sono quelli per ottenere tali informazioni.

Triple

Una tripla di tre tipi generici A,B,C. E' usata per agevolare la restituzione delle informazioni da ConnectionsServerSide a Execution.

Utente

Vi sono memorizzate tutte le informazioni relative ad un utente. Si noti che per non replicare informazioni, gli elenchi di documenti creati, documenti ai quali l'utente è invitato alla modifica e documenti ai quali l'utente è invitato ma non gli è ancora stato notificato non contengono il documento vero e proprio, ma solo il nome.

5 Istruzioni per l'esecuzione

5.1 Assunzioni sull'interfaccia Client

L'interfaccia del Client segue quanto specificato nell'appendice A del testo del progetto, l'unica modifica è relativa al comando show.

L'attivazione della SHOW DOC prevede l'introduzione di -1 come ultimo parametro per distinguerla dall'interfaccia della SHOW DOC SEC

Es. *turing show <doc > <sec >* richiede la visualizzazione della sezione <sec> del documento <doc>

turing show <doc > <-1> richiede la visualizzazione di tutte le sezioni del documento <doc>

5.2 Istruzioni per la compilazione

Verificare che nel PATH dell'ambiente sia presente il percorso per l'eseguibile javac.exe

Per la compilazione, spostarsi nella directory Turing\Server, lanciare da riga di comando Windows:

*javac *.java*

Ripetere analogamente l'operazione nella directory Turing\Client.

5.2 Esecuzione dello script di test

Prima di iniziare il test è necessario verificare che le porte utilizzate dal protocollo TCP e UDP nell'applicazione risultino non occupate.

Eseguire il test da linea di comando Windows :

netstat –an

Se l' indirizzo 127.0.0.1 risulta occupato sulla porta 10001 su TCP (porta utilizzata dal socket listener del Server) è necessario procedere alla modifica della classe MainClassTuringServer e cambiare il valore della variabile SERVICE_PORT con un numero di porta libero.

Se nel range di indirizzi da 239.0.0.0 a 239.0.0.255 risulta occupata la porta 12000 su UDP (porta utilizzata per le comunicazioni chat) è necessario modificare la classe MainClassTuringServer e cambiare il valore della variabile CHAT_PORT con un numero di porta libero.

Dopo avere effettuato le verifiche sulle porte, lanciare in sequenza i seguenti script:

demo1.bat

demo2.bat

demo3.bat

Ogni script ha come prima istruzione la kill di processi Client e Server dell'applicazione Turing che sono stati eventualmente attivati in precedenza.

La cartella ClientInput contiene i file di input per ogni script mentre nella cartella log sono creati i file con lo standard output ed error del Client e del Server rispettivamente.

Si noti che nel caso di esecuzione manuale dell'applicazione è necessario verificare che nella cartella Turing\Server non siano presenti directory con il nome del documento che si vuole creare nella corrente prova. In tal caso bisogna provvedere manualmente alla rimozione della directory e dei file sottostanti.