UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

SISTEMI OPERATIVI E LABORATORIO

2017/2018

Chatterbox

Autore: *Stefano Torneo*

Docente*:* Massimo Torquati

Matricola*:* 545261

Corso*:* B

13 Gennaio 2019

Indice

1. Obiettivo 3
2. Strutture dati 3
   2.1 Hash per gli utenti registrati 3
   2.2 Hash per i gruppi 3
   2.3 Lista per gli utenti online 4
   2.4 Coda concorrente per le richieste dei client 4
3. Gestione della memoria 5
4. Accesso ai files 5
5. Gestione dei segnali 5
6. Statistiche 6
7. Struttura del codice 6
8. Struttura del server 7

8.1 Comunicazione di ritorno tra listener e worker 7

8.2 Protocollo di terminazione 7

9. Test e macchina utilizzata 8

1. Obiettivo

L'obiettivo del progetto è quello di realizzare un server che risponda alle richieste di uno o più client. Il codice che implementa il client è stato fornito dal docente.

2. Strutture dati

Le strutture dati utilizzate dal lato server sono:

- Hash per la gestione degli utenti registrati e della loro history.
- Hash per la gestione dei gruppi.
- Lista per la gestione degli utenti online.
- Coda concorrente per la gestione delle richieste dei client.

2.1 Hash per gli utenti registrati

Ho scelto di utilizzare una tabella hash per la gestione degli utenti, per avere un accesso più veloce alle informazioni.
Ogni utente registrato verrà inserito nella tabella hash, memorizzandone il nickname.
La struttura memorizza al suo interno anche un puntatore alla history, ovvero una struttura dati che memorizza i messaggi dell'utente, da chi sono stati inviati e il tipo del messaggio (testuale o file). La history è gestita come un array circolare, quindi provvista di un puntatore in testa e uno in coda.
La struttura hash ha tra le operazioni implementate:

- Inserimento di un utente.
- Cancellazione di un utente.
- Ricerca di un utente.
- Inserimento di un messaggio nella history di un utente.
- Inserimento di un messaggio nella history di più utenti.
- Inserimento di un messaggio nella history di uno o più utenti appartenenti ad un gruppo.
- Visualizzazione dei messaggi presenti nella history di un utente.

Le funzioni sono eseguite in mutua-esclusione, suddividendo logicamente la hash in zone, dove il numero di zone corrisponde al numero di thread nel pool, specificato nei file di configurazione. In questo modo un thread bloccherà una sola zona e questo implica che è possibile avere più accessi in contemporanea da thread diversi in parti di struttura differenti.
Per la lunghezza della struttura hash, non essendo specificata nei file di configurazione, ho scelto essere pari a 1024.

2.2 Hash per i gruppi

Anche per la gestione dei gruppi ho scelto di implementare una hash.
All'interno della struttura memorizzo il nome del gruppo, la lista di utenti e il numero di iscritti ad un gruppo.
La struttura hash ha tra le operazioni implementate:

- Creazione di un gruppo.

- Cancellazione di un gruppo.
- Ricerca di un gruppo.
- Inserimento di un utente in un gruppo.
- Ricerca di un utente in un gruppo.
- Cancellazione di un utente da un gruppo.

Anche questa struttura è divisa in zone per la gestione della mutua-esclusione.

Non essendo specificata nei file di configurazione, ho deciso di impostare un numero massimo di utenti registrabili in un gruppo (32) e una dimensione della struttura (1024).

Politiche Gruppi:

1. Solo l'utente che ha creato il gruppo avrà il diritto di cancellare il gruppo, se il creatore abbandona il gruppo allora questo diritto passa al secondo iscritto, e così via.
2. Se c'è un solo iscritto e questo si deregistra dal gruppo, allora quel gruppo viene eliminato.

2.3 Lista per gli utenti online

Per la gestione degli utenti online ho deciso di utilizzare una lista, in modo tale che quando un utente invia una richiesta di connessione, se viene accettata, allora l'utente viene inserito in tale struttura. Ogni volta che si invia un messaggio si controlla se il destinatario è online, in tal caso si prova a consegnare il messaggio.
Un utente rimarrà in questa struttura fino al momento della sua disconnessione. Per ogni utente si memorizzano:

- descrittore
- nome
- variabile di mutua-esclusione.

La struttura offre le seguenti funzionalità:

- Inserimento di un utente nella lista.
- Cancellazione di un utente dalla lista.
- Ricerca di un utente tramite descrittore.
- Ricerca del descrittore tramite nome dell'utente.
- Invio in mutua-esclusione dei dati.
- Visualizzazione della lista degli utenti online.

Per la gestione della mutua-esclusione viene utilizzata una variabile di mutex globale, utilizzata in ogni funzione.
Inoltre, per le operazioni di invio di messaggi, history e lista utenti online, ho ritenuto opportuno introdurre un'ulteriore variabile di mutex, una per ogni utente. Quando il server deve inviare un'informazione riguardante uno specifico utente, bisogna estrarre il suo descrittore e prendere la lock sulla variabile di mutua-esclusione dell'utente garantendo che un solo utente alla volta possa inviargli messaggi.

2.4 Coda concorrente per le richieste dei client

Questa struttura è stata pensata per gestire le richieste da parte dei client.
Quando il server riceve le richieste, immette in una coda il descrittore del client che ha fatto quella richiesta, in modo tale che i thread che gestiscono le richieste possono accedere alla coda, in mutua-esclusione, ed effettuare le operazioni necessarie affinchè la richiesta venga soddisfatta.
Le due operazioni implementate sono:

- Inserimento di un descrittore in coda: quando il server riceve una richiesta.
- Cancellazione di un descrittore in coda: quando un thread estrae il descrittore dalla coda per servire la richiesta.

3. Gestione della memoria

Ho deciso di allocare dinamicamente le strutture e le variabili al suo interno in modo da non sprecare memoria.
Ad ogni allocazione corrisponde una deallocazione opportuna.
Ogni struttura viene distrutta dal server prima di terminare, in modo tale da evitare di avere memory leaks.
Tutte le allocazioni sono controllate da una macro che, nel caso in cui non sia possibile allocare, fa terminare il server.

4. Accesso ai files

Il server interagisce con diversi file durante la sua esecuzione:

- inizialmente apre e legge il file di configurazione passatogli come parametro;
- successivamente permette lo scambio di files tra i client e quindi provvede a creare/aprire e scrivere/leggere su file;
- inoltre, interagisce con il file delle statistiche, su cui scrive le informazioni calcolate nelle varie funzioni.

5. Gestione dei segnali

Il server può ricevere dall'utente essenzialmente due tipi di segnali:

- Segnali di terminazione
- Segnale SIGUSR1

In entrambi i casi, ho deciso di gestire i segnali tramite un signal handler.
Per i segnali di terminazione, ho installato lo stesso signal handler. Quando arriva uno di questi segnali, il suo compito è quello di andare a settare una variabile apposita, che verrà controllata periodicamente dal Listener il quale inizierà il protocollo di terminazione.
Per il segnale SIGUSR1, vale lo stesso principio con la differenza che quando il suo signal handler andrà a settare una variabile apposita, il Listener salverà le statistiche in un file, il cui percorso è specificato nei parametri di configurazione. Inoltre, viene ignorato il segnale SIGPIPE.
Per non avere problemi in fase di inizializzazione, maschero tutti i segnali finchè i gestori non sono stati installati, e tolgo la maschera dopo aver impostato i gestori.

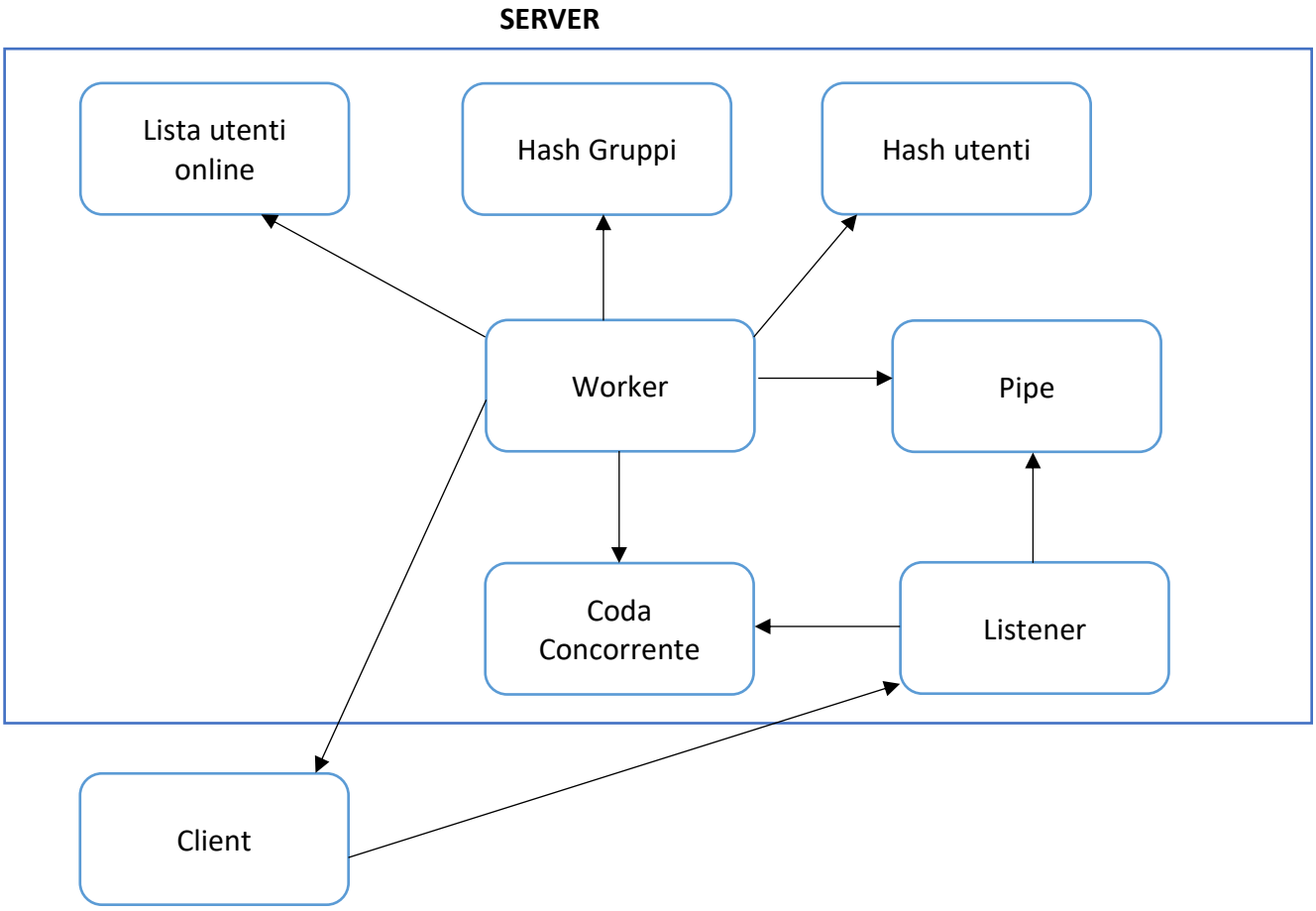
6. Statistiche

Per manipolare correttamente le variabili della struttura delle statistiche, fornita dal docente, ho pensato di aggiungere una variabile “consegnato” nella history di ogni utente in modo tale che quando un utente legge dei messaggi dalla propria history e questi sono stati già letti, non vengono conteggiati più volte.
Inoltre, ho usato una variabile di mutua-esclusione per le statistiche, quindi prendo la lock sulla variabile prima di ogni manipolazione delle statistiche e la rilascio una volta finito.

7. Struttura del codice

Il codice è stato diviso in più moduli, tra i quali:

- Modulo per l’hash degli utenti registrati
- Modulo per l’hash e le funzionalità dei gruppi
- Modulo per la lista degli utenti online
- Modulo per la coda concorrente
- Modulo per le funzioni di connessione
- Server chatty
- Parser
- Script



Comunicazione:

- **Client->Server**: tramite modulo per le funzioni di connessione.
- **Client->Listener**: un client richiede connessione al server.
- **Listener->Coda Concorrente**: il listener aggiunge una richiesta nella coda dei descrittori.
- **Listener->Pipe**: il listener legge dalla pipe un descrittore e lo inserisce nella maschera di descrittore pronti.
- **Worker->Pipe**: un worker inserisce il descrittore nella pipe dopo aver servito la richiesta.
- **Worker->Hash Utenti**: richiesta di invio di un messaggio.
- **Worker->Gruppi**: creazione, registrazione e cancellazione di un gruppo o di un utente.
- **Worker->Coda Concorrente**: un worker estrae il primo descrittore presente nella coda.
- **Worker->Lista online**: un worker inserisce il descrittore e il nome dell'utente a seguito di una connessione.

8. Struttura del server

Il server è strutturato in modo da avere:

- Un unico thread **listener**: che ha il compito di ascoltare nuove connessioni, accettarle, registrarle in una maschera di bit e gestire nuovi eventi, come l'arrivo di un segnale e/o richieste da parte di client già connessi. Il listener è basato sulla funzione select, che gestisce richieste da parte di più client. Alla select è associato un timer in modo tale che allo scadere la select possa vedere se ci sono descrittori di cui bisogna servire la richiesta o se il signal handler ha intercettato dei segnali. Quando un nuovo client fa richiesta, la select mette il descrittore nella sua maschera di descrittori, e quando essa ritorna, dice quali descrittori sono pronti, ovvero che una operazione di lettura/scrittura su quel descrittore non si blocca. Quando il listener si accorge che il descrittore è pronto, lo estrae dal set della select e lo inserisce in coda pronti.
- E più **workers**, che hanno il compito di prelevare un descrittore pronto dalla coda concorrente e mettersi in ascolto di una richiesta su quel descrittore. Ogni thread del pool soddisfa al più una richiesta, quindi richieste distinte dello stesso client non necessariamente vengono gestite dallo stesso thread. Dopo aver ricevuto ed eseguito la richiesta inizia una comunicazione all'indietro con il thread listener, ovvero il worker comunica al listener il descrittore appena servito tramite pipe. Al termine, il thread worker controlla se vi sono altri descrittori di connessione pronti in coda altrimenti si sospende.

8.1 Comunicazione di ritorno tra listener e worker

Il worker, dopo aver servito una richiesta, scrive il descrittore in una pipe. Il descrittore della pipe è inizialmente registrato nella select. Quando la select ritorna, se un worker ha scritto nella pipe, quel descrittore sarà settato. Quando si scandisce il set, se quel descrittore è settato ed è quello della pipe, allora si legge dalla pipe e il descrittore letto viene registrato di nuovo nella select. In questo modo il client potrà fare altre richieste che possono essere così gestite da un qualsiasi thread del pool.

8.2 Protocollo di terminazione

Nel listener, viene impostato un timer per la select, in modo tale che ogni volta che scade va a vedere se una particolare variabile globale è stata settata, la quale indica che è stato ricevuto un segnale. Quando viene ricevuto un segnale di terminazione allora viene aggiunto '-1' alla coda delle richieste, dopo di che il listener termina. A questo punto il main aspetta che tutti i workers terminano la loro esecuzione. Il primo worker estrae '-1' e risveglia tutti i workers sospesi.
Una volta risvegliati i workers si accorgono che l'elemento estratto è '-1', quindi smettono di servire richieste e terminano la loro esecuzione.
A questo punto il main può proseguire e deallocare tutte le strutture dati usate dal server quali hash utenti registrati, hash gruppi, coda descrittori e lista utenti online.

9. Test e macchina utilizzata

Il progetto è stato sviluppato in una macchina virtuale in cui è stata installata la versione di Ubuntu fornita dal docente nella pagina del corso, e testato anche in una macchina con Ubuntu 16.04 lts e nelle macchine del laboratorio.
Nel makefile è stato aggiunto del codice per poter eseguire lo script che testa le funzionalità che riguardano i gruppi, per far partire il test basta digitare "make test6".