Gaspare Ferraro matricola 520549 corso B

chatterbox

Progetto per il modulo
di laboratorio SOL 2017

Relazione del progetto

Strutture dati

Iniziamo descrivendo le strutture dati utilizzate nel progetto, dal punto di vista delle funzionalità offerte, per poi analizzarne il loro utilizzo all’interno del progetto.

HashMap, leggera implementazione di una mappa, vista a lezione, che fornisce operazioni di creazione e distruzione di mappe, inserimento e rimozioni di coppie chiave-valori in base alle chiavi e ricerca di valori per chiave.

Queue, coda implementata mediante array circolare di dimensione finita, decisa alla creazione, che fornisce operazioni mutuamente esclusive di pop e push di interi oltre a costruttore e distruttore della struttura dati in memoria.

LinkedList, lista implementata mediante nodi collegati a catena che fornisce la possibilità di gestire informazioni generiche e di poter decidere la dimensione massima, eventualmente infinita, del numero di elementi. Anche in questo caso vengono fornite le operazioni di pop e push, costruttore e distruttore della struttura dati in memoria.

Op, le operazioni fornite dal server vengono codificate con il client mediante interi predefiniti associati.

Message, un messaggio scambiato tra client e server viene codificato come una coppia header-data, la parte header specifica il richiedente e l’operazione da svolgere, la parte data specifica il destinatario dell’operazione e il buffer di dati associato.

ServerConfiguration, struttura che contiene le configurazioni necessarie del server, lette dal file specificato all’avvio.

Statistics, struttura che contiene le statistiche raccolta dal server riguardo le operazioni effettuate, in termini di messaggi, file, errori e utenti.

User, identifica un utente registrato sul server del quale vengono memorizzati il nickname che lo identifica, la cronologia dei messaggi pendenti ancora da inviare e un descrittore della connessione se l'utente risulta colleggato.

Group, un gruppo viene identificato sul server dal suo nome e dalla lista dei membri che sono iscritti a tale gruppo.

Le ultime due strutture, più complesse e basate sulle precedenti, fornisco metodi per gestire le due entità principali del server.

UserManager, gestisce gli utenti offrendo operazioni di registrazione/cancellazione, connessione/disconnessione, invio di messaggi pendenti, elencazione degli utenti collegati oltre che a costruttore e distruttore delle strutture.

GroupManager, gestisce i gruppi offrendo operazioni di creazione/cancellazione di nuovi gruppi, registrazione/deregistrazione di utenti a gruppi e operazioni di ricerca e controllo su utenti e gruppi.

Gestione memoria

Particolare attenzione è stata rivolta alla gestione della memoria dinamica: dopo ogni operazione di allocazione è stata eseguita un'operazione di pulizia, al fine di evitare accidentali operazioni su memoria *sporca*, ricordandosi inoltre di de-allocare la memoria richiesta quando questa non è più utilizzata.
Inoltre il server subito prima di terminare effettua le ultime operazioni di pulizia, usando tra l'altro le funzioni di distruzione delle strutture dati precedentemente descritte.

Accesso ai files

Il server nel corso della sua esecuzione interagisce con diversi files, in particolare all'avvio legge le configurazioni dal file specificato e alla ricezione del segnale SIGUSR1 appende su un file le statistiche raccolte.
Per ogni file scambiato tra gli utenti ne viene salvata una copia nella cartella specificata nella configurazioni, omettendo eventuali percorsi nel nome del file ricevuto.
Tutte le operazioni sui file vengono svolte usando le funzioni standard di lettura e scrittura bufferizzate.

Divisione in files

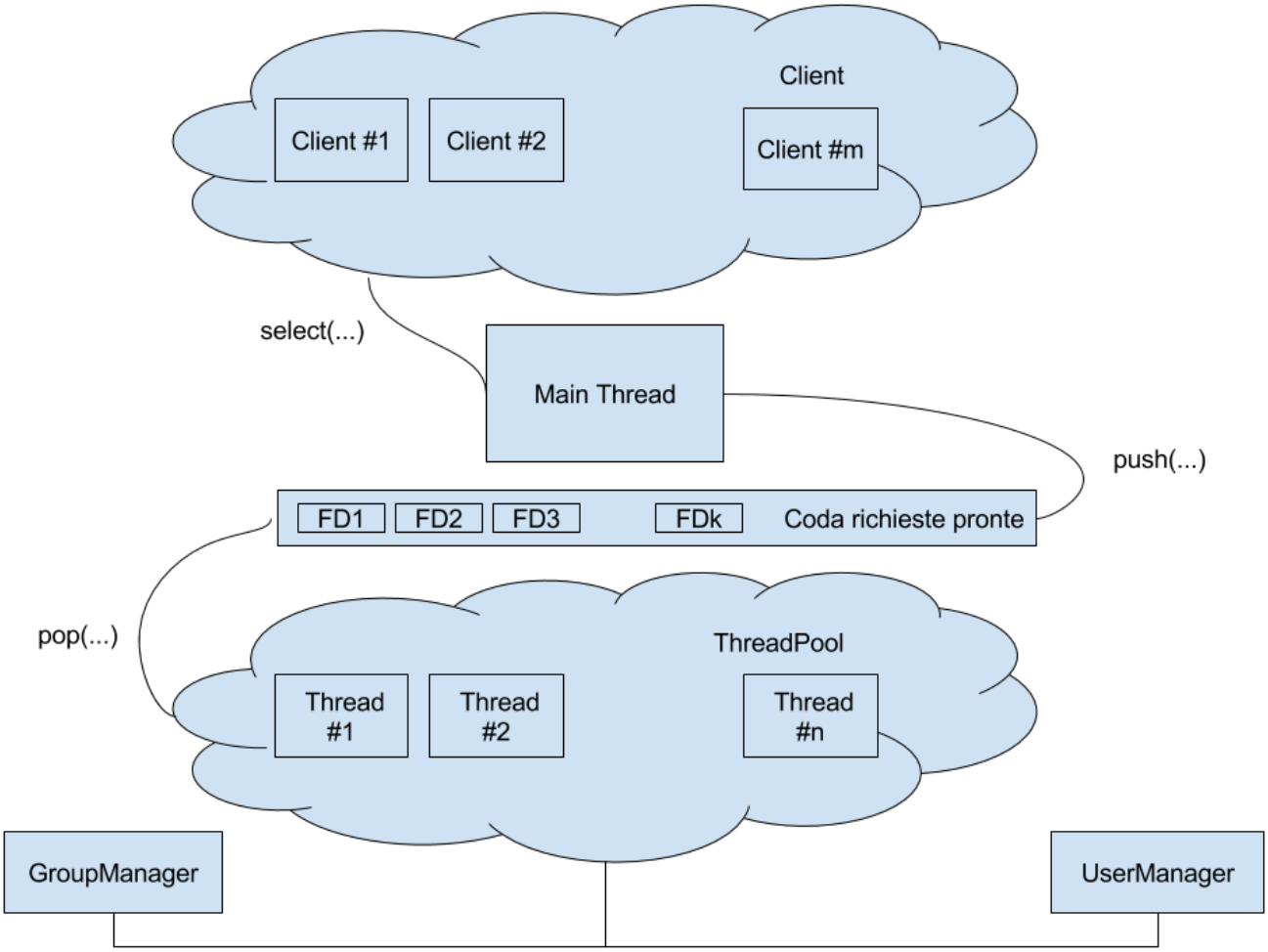
Per migliorare la gestione dei singoli moduli, il progetto è stato suddiviso su più files, ognuno dei quali dichiara e implementa funzionalità vicine tra loro.
In particolare sono stati scritti file per:

- gestione delle configurazioni
- protocollo di comunicazione
- gestione degli utenti
- gestione dei gruppi
- hashmap
- lista linkata
- coda circolare
- operazioni e messaggi

Il file sorgente *chatty.c* si preoccupa di raccogliere tutti i moduli per fornire le funzionalità richieste.

Architettura del server

L’architettura implementata dal server è stata scelta al fine di suddividere il carico di lavoro tra i diversi thread attivati nel modo più equilibrato possibile, ottenendo quindi performance elevate sfruttando al meglio il parallelismo della ThreadPool.
Il Thread principale all’avvio si preoccupa di caricare le configurazioni, allocare i gestori e le strutture dati utilizzate, creare e associare al path definito il socket per la comunicazione con i diversi client e, infine, la creazione dei Thread della ThreadPool che si metteranno in attesa delle richieste da servire.



Lo schema sopra descrive graficamente a grandi linee come avvengono le comunicazioni tra le varie parti del server.
Il Thread principale si mette in ascolto, tramite una select, sia di nuovi client che vogliono unirsi alla chat sia di client già connessi che non stanno effettuando operazioni.
Quando uno tra questi vuole richiedere un'operazione, viene inserito nella coda delle richieste pronte e rimosso dai client tra i quali viene chiamata la select.
La coda delle richieste pronte viene smaltita dai Thread della ThreadPool, tra quelli non impegnati a svolgere altre operazioni, che prendono in carico tali richieste.
Al termine dell'operazione si reinseriscono i client serviti tra quelli in ascolto dal Thread principale.
Nel caso di disconnessione di un client la sua connessione viene chiusa e rimossa da quelle in ascolto.

Concorrenza

La concorrenza tra i Thread è stata gestita in maniera da non introdurre overhead nell'esecuzione del server sia in termini di tempo che di memoria richiesta.
Come primo accorgimento è stata introdotta una variabile di condizione, alla coda delle richieste pronte, in modo da notificare i Thread della ThreadPool che sono in attesa di svolgere dei compiti, quando viene inserito un nuovo client da servire; tale accorgimento può essere visto come un problema tipo Produttore-Consumatore (un solo produttore, il thread principale, tanti consumatori, i Thread della ThreadPool).
Inoltre si è preferito introdurre un solo mutex per ogni struttura principale, invece che per ogni singolo utente/gruppo, in modo da non avere un numero eccessivo di mutex che, oltre ad introdurre overhead, risultano poco utili in quanto ogni client viene gestito da un solo Thread alla volta e ogni operazione svolta in mutua esclusione viene eseguita in breve tempo.
Un miglioramento aggiuntivo, nel caso si vogliano migliorare le prestazioni all'aumento del numero degli utenti, può essere quello di introduzione mutex di tipo ReadWrite che permettono a diversi lettori l'accesso in parallelo ma in mutua esclusione con i singoli scrittori.

Debugging e verifica

Per il debugging sono stati utilizzati sia delle classiche stampe per la verifica del flusso di esecuzione e analisi del contenuto della variabili, sia software quali gdb e valgrind per l’analisi della memoria dinamica. Il progetto è stato sviluppato e testato su una macchina con linux distribuzione Arch, i test forniti sono stati effettuati e passati sia sulla macchina di sviluppo sia su un’altra macchina con linux distribuzione Ubuntu 16.04 LTS.