Documentazione OSproject

Chat Multicanale

A.A. 15/16
Appello di giugno 2016.

Progetto di Sistemi operativi del corso di laurea triennale presso “La Sapienza Università Roma”, sviluppata dagli studenti:

- Jacopo Carlini matricola 1598460
- Roberto Dessì matricola 1619524

Indice

Introduzione

Il progetto della chat multicanale fornisce un servizio di messaggistica istantanea all'utente.
Connettendosi, attraverso l'invio di specifici comandi, è possibile interagire con il server residente su una diversa macchina.
L'utente connesso può decidere se creare un nuovo canale di comunicazione, effettuare l'operazione di "join" ad un canale esistente oppure richiedere la lista dei canali fino ad ora creati.
Una volta eseguite una delle due operazioni, ed essendo a tutti gli effetti all'interno di un canale è possibile chattare liberamente con tutti i partecipanti.
Una volta all'interno del canale, è possibile inoltre uscirne oppure eliminarlo qualora l'utente che intenda farlo ne sia il creatore.

- I comandi accettati appena viene instaurata la connessione al server sono i seguenti:

NOME COMANDO	SEMANTICA
/show	viene mostrata la lista di tutti i canali al momento attivi
/create <nome canale>	viene creato un nuovo canale con il nome indicato
/join <nome canale>	eseguito il join al canale indicato, se presente

- I comandi accettati dal server una volta dentro il canale sono i seguenti:

NOME COMANDO	SEMANTICA
<*>	Il testo inserito viene automaticamente inoltrato ad ogni partecipante al canale
/quit	l'utente decide di uscire dal canale
/delete	il canale viene cancellato qualora l'utente che abbia inoltrato il comando /delete ne fosse il proprietario

Specifiche

Realizzazione di un servizio "chat" via internet offerto tramite server che gestisce un insieme di processi client (residenti, in generale, su macchine diverse). Il server deve essere in grado di acquisire in input da ogni client una linea alla volta e inviarla a tutti gli altri client attualmente presenti nella chat.
Ogni client potrà decidere se creare un nuovo canale di conversazione, effettuare il join ad un canale già esistente, lasciare il canale, o chiudere il canale (solo nel caso che l'abbia istanziato).

Si precisa che la specifica richiede la realizzazione del software sia per l'applicazione client che per l'applicazione server.

Per progetti misti Unix/Windows e' a scelta quale delle due applicazioni sviluppare per uno dei due sistemi.

Architettura

L'architettura della piattaforma ha una struttura di tipo client-server.

Il client ha una struttura interna che utilizza due thread, uno con lo scopo di inviare i messaggi al server, l'altro con lo scopo di ricevere e stampare a schermo quello che è stato ricevuto.

Il server invece, è sviluppato con una struttura di tipo multithread. Il thread principale resta perennemente in ascolto di eventuali connessione da accettare.
Una volta ricevuta una richiesta di connessione viene eseguito lo spawn di un nuovo thread che si occuperà di gestire da questo momento in poi la connessione il client appena collegato.

Un'immagine esplicativa dell'architettura della piattaforma è mostrata nella Figura 1.

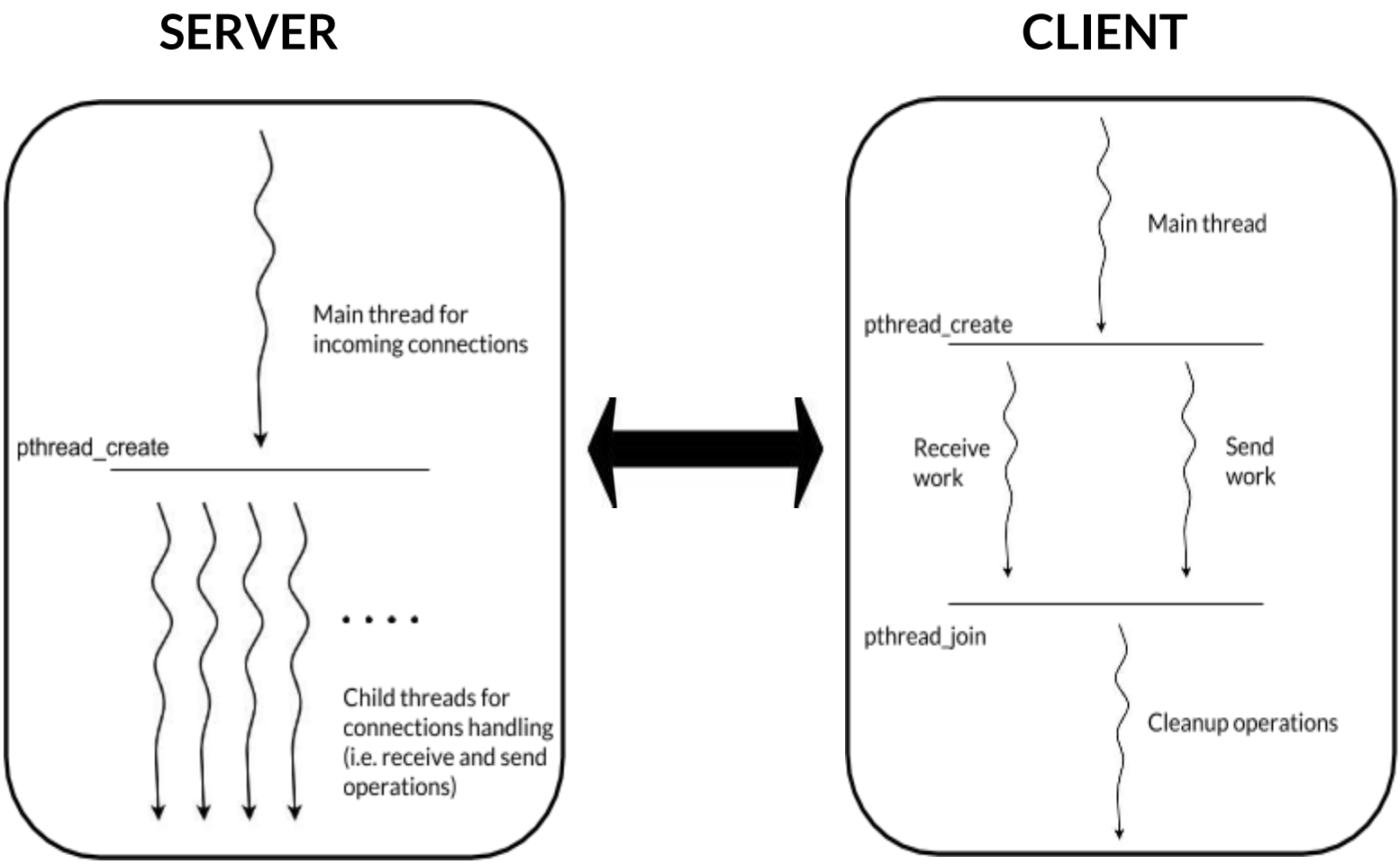


Fig. 1: Immagine dell'architettura della piattaforma

È stato scelto un approccio multithread invece che multiprocesso per una maggiore ottimizzazione delle prestazioni.
Nel nostro caso infatti, un approccio multiprocesso avrebbe appesantito la creazione di un nuovo "gestore" della connessione in entrata.
Il nuovo processo avrebbe inoltre dovuto accedere a strutture dati condivise create dal processo padre, oltre a copiarne lo spazio di memoria alla creazione.
Si è quindi deciso di sfruttare il più possibile la condivisione di memoria garantita da un approccio multithread, garantendo la sincronizzazione per l'accesso a strutture dati condivise grazie all'utilizzo di named semaphore.
Si è scelto l'utilizzo di quest'ultimi rispetto agli unnamed semaphore per il fatto che parte dello sviluppo del progetto è avvenuto su piattaforma Mac e sistema operativo OS X e quest'ultimo non supporta l'utilizzo dei semafori unnamed.
Per la sincronizzazioni fra i vari thread gestori delle connesioni riguardo la chiusura di un canale della chat si è optato per l'utilizzo di code di messaggi.
Ogni thread gestore possiede quindi la proprio coda di messaggi sulla quale riceve comunicazione dell'eventuale chiusura del canale a cui il client è potenzialmente connesso (messaggio di tipo 1). Messaggio di catch del segnale Sigterm e quindi necessità di cleanup di buffer e strutture dati (messaggio di tipo 3). Il thread gestore, per necessità di sincronizzazione invia quindi un messaggio di ack (messaggio di tipo 2).

Per maggiori informazioni sull'incompatibilità dei semafori unnamed su sistema operativo OS X si veda:

- https://developer.apple.com/library/ios/documentation/System/Conceptual/ManPages_iPhoneOS/man2/sem_open.2.html
- http://heldercorreia.com/blog/semaphores-in-mac-os-x
- http://stackoverflow.com/questions/1413785/sem-init-on-os-x

Codice sorgente

Di seguito il codice sorgente della piattaforma e una breve descrizione di ogni file e del suo contenuto.

File sorgente server:

- server.c
  questo file contiene la struttura centrale del server. Sono presenti i metodi e le funzioni che si occupano della gestione dei segnali. Vengono create le strutture dati per la gestione dei canali di comunicazione (array di canali) e il semaforo necessario ad accedervi. Vengone eseguite le funzione di socket(), bind(), listen() e all'interno di un ciclo while(1) la funzione di accept(). Viene successivamente creato (sempre all'interno del ciclo while) un thread che si occuperà di gestire la connessione e gli vengono quindi passati come parametro tutte le strutture dati necessarie.

- thread.c
  questo file contiene tutto il lavoro svolto da ogni thread creato per gestire una connessione. Viene infatti istanziata la coda di messsaggi proprio di ogni thread. Ci si metterà quindi in continuo ascolto tramite la funzione ricevi() di eventuali messaggi da parte del client.

  Tali messaggi possono essere:

  - /create <nome canale>: viene creato un nuovo canale e le strutture dati necessarie alla sua gestione come la lista dei partecipanti al canale e il semafoto per accedervi
  - /join <nome canale>: il client che ha effettuato la richiesta viene aggiunto al canale. Viene quindi aggiornata in concorrenza (sincronizzazzione avvenuta attraverso il semaforo proprio di ogni canale) la struttura dati del canale
  - /quit: questo comando comporta l'uscita dal canale da parte del client che l'ha inviato. Esso viene quindi eliminato dalla lista dei partecipanti
  - /delete: eliminazione del canale e di tutte le strutture dati ad esso collegate. Affinchè il canale sia effettivamente eliminato, solo il suo proprietario può inviare tale messaggio. In caso di effettiva eliminazione, viene mandato un messaggio attraverso la coda di messaggi ad ogni partecipante al canale per informare dell'imminente eliminazione e poter quindi eseguire le dovute funzioni. Vengo alla fine eliminate tutte le

strutture dati precedentemante utilizzate per il canale (i.e. semaforo del canale, nome del canale, lista dei partecipanti)

- /show: viene richiesta al server e quindi inviata la lista dei canali che sono attualmente disponibili per poter effettuare una /join

Ogni altro messaggio verrà inoltrato a tutti i partecipanti al canale a cui il client è connesso. Nel caso non sia connesso a nessun canale e invia un messaggio che non sia uno dei sopracitati verrano automaticamente inviate in risposta le istruzioni di funzionamento.

- thread.h

  File contenente le strutture dati necessarie allo svolgimento delle funzioni del thread gestore e intestazione dei metodi utilizzati in thread.c

- thread_util.c

  In questo file sono presenti funzioni di supporto che il thread dovrà svolgere come:
  - l'invio e la ricezione di un messaggio su socket
  - una funzione di supporto alla lettura di un messaggio dalla coda
  - una funzione che permette a un thread di uscire da un canale aggiornando in maniera contestuale le opportune strutture dati
  - una funzione che prende il nome del nuovo canale da creare
  - due funzione di print su terminale di ogni canale e delle sue informazioni (i.e. numero di partecipanti e nome del canele ecc.)

- thread_util.h

  file header contente le intestazioni dei metodi utilizzati in thread_util.c

- common.h

  file header contenente due error helper per gestire la chiusura anomala del server. Sono inoltre presenti variabili globali utilizzate in molteplici file.

- log.c

  sono qui contenuti i metodi necessari per effettuare tutte le differenti operazioni di audit sul file di log.

- log.h
  Sono qui presenti le intestazioni dei metodi utilizzati nel file log.c

- Makefile
  Makefile utilizzato per la compilazione (comando make da terminale), l'esecuzione (make run) e l'eliminazione degli eseguibili creati (make clean)

File sorgente client:

- client_windows.cpp
  Questo file contiene la struttura del client Windows. Vengono gestiti i segnali, acquisiti i comandi inviati dalla command line, viene inizializzata la connessione con il server e, in caso di successo viene effettuato lo spawn di due thread. Uno è dedicato alla ricezione e l'altro all'invio di messaggi. Si effettua quindi un'operazione di join e infine una funzione di cleanup.

- common.h
  Questo file header contiene due errore helper da utilizzare in client_windows in caso di chiusura anomala.

Funzionamento

Server:

Per l'esecuzione del server è sufficiente aprire il terminale, posizionarsi nella cartella contente i file del codice sorgente e digitare:

1. make
2. make run

Il server sarà quindi in ascolto sulla porta 2016.

Client:

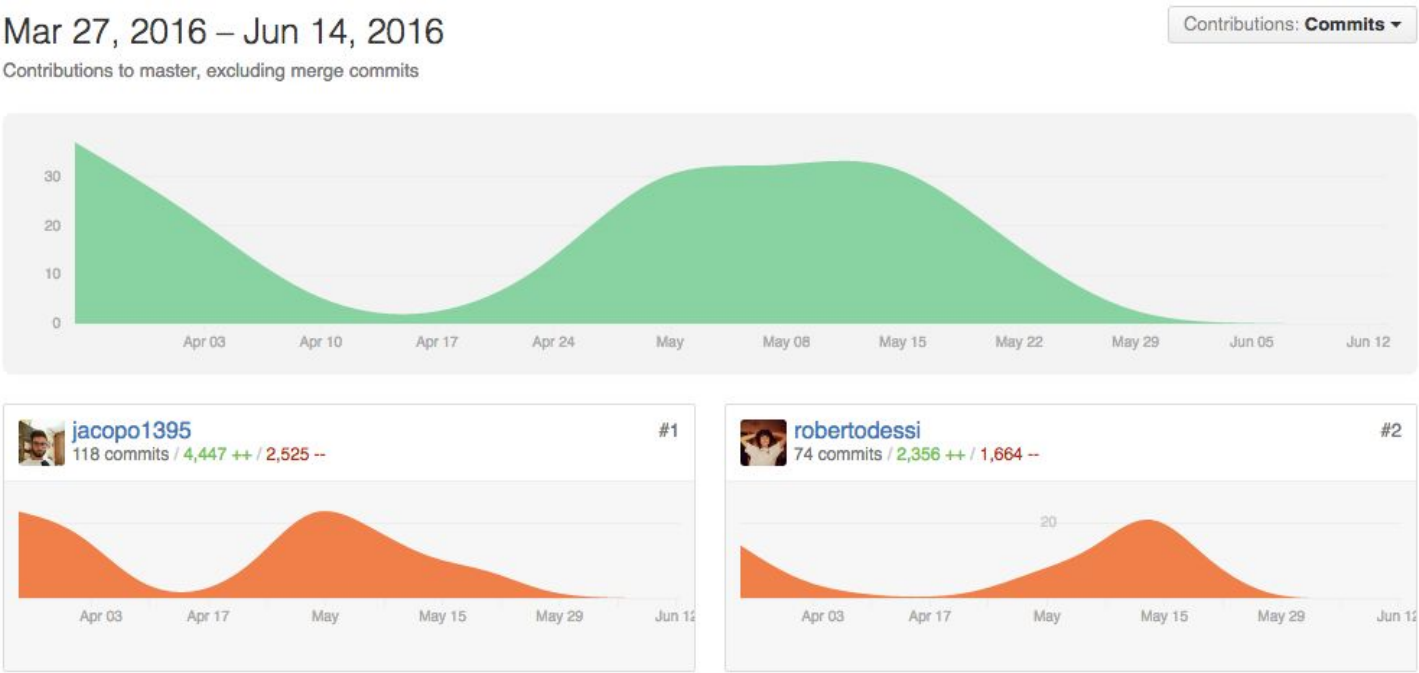
Per utilizzare l'applicativo client è sufficiente posizionarsi nella cartella contente i file del codice sorgente ed eseguire le seguenti istruzioni:

1. compilare il file
2. dal prompt dei comandi digitare:
   ./client -a (indirizzo server) -p (porta del server) [-h]

Se presente, il suffisso -h, indica la richiesta di invio di informazioni sulla sintassi dei comandi.

Distribuzione e carico del lavoro

Vengono presentate due immagini che rappresentano il carico e la distribuzione del lavoro ripartito fra i due membri del gruppo.
Nelle immagini (Fig. 2 e Fig. 3) vengono mostrati dei dati raccolti dal sito di github del progetto (https://github.com/robertodessi/OSproject)



Fig, 2: Distribuzione dei commit nel tempo. Fonte: https://github.com/robertodessi/OSproject/graphs/contributors

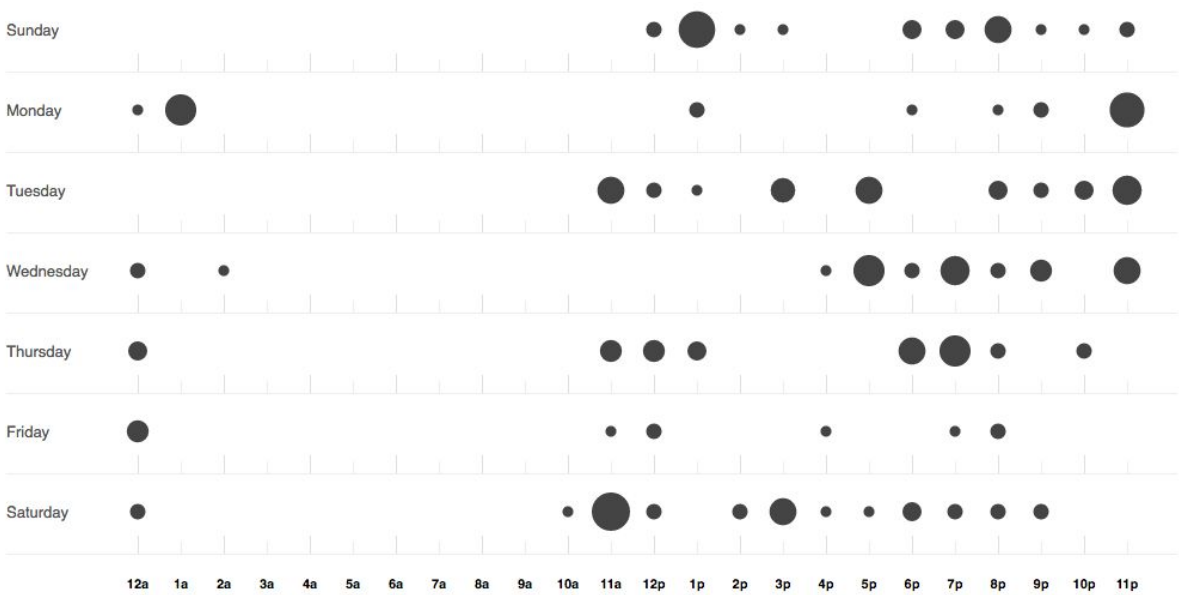


Fig. 3: Frequenza settimanale dei commit. Fonte: https://github.com/robertodessi/OSproject/graphs/punch-card

Fonti

La maggiore fonte di informazioni per lo sviluppo del progetto sono state reperite dalla documentazione ufficiale sullo sviluppo in ambiente UNIX, dal sito http://stackoverflow.com/ e dal materiale del sito di del corso di "Sistemi Operativi" http://www.dis.uniroma1.it/~quaglia/DIDATTICA/SO-ORD-2013/ e le dispense del corso di "Sistemi di Calcolo" http://www.dis.uniroma1.it/~sc/wikka.php?wakka=Materiale1415

Importanti riferimenti sono stati inoltre le seguenti pagine web:

- http://man7.org/linux/man-pages/man2/ (gestione coda di messaggi)
- http://man7.org/linux/man-pages/man7/ (gestione segnali Linux, multithread e funzionamento code di messaggi)
- http://www.linuxquestions.org/questions/programming-9/prevent-process-crashing-on-thread-crash-870712/ (forum con domande sulla gestione dei segnali linux)
- https://social.msdn.microsoft.com/Forums/en-US/ac626c1d-9449-4f4d-804f-115a1779b284/sigpipe-signal?forum=netfxnetcom (forum con domande sulla gestione segnali windows)
- https://tangentsoft.net/wskfaq/articles/bsd-compatibility.html (gestione segnali windows)

Riferimenti

Il link github al progetto è il seguente: https://github.com/robertodessi/OSproject

Le email alle quali è possibile contattarci:

- Jacopo -> jack1395@hotmail.it
- Roberto -> roberto.dessi11@gmail.com

I nostri profili github:

- Jacopo -> https://github.com/robertodessi
- Roberto -> https://github.com/jacopo1395