ChatterBox

Descrizione dell'implementazione del server

Luca Canessa (516639)

03-04-2018

INDICE

1. Introduzione

Questo documento è stato redatto per delineare l'implementazione del server. Tutta la progettazione si è basata sulle direttive predefinite. Il server è stato creato, come da richiesta, per soddisfare istanze concorrenti provenienti da diversi client.

Ogni parte del codice è stata documentata e trascritta in un file pdf, contenuto nella directory "docs", utilizzando il tool *"doxygen"*, nel quale è stato chiarito il contenuto di ogni singolo file e spiegata ogni funzione e ogni possibile struttura dati contenuta in esso.

1.1 Descrizione Generale

Il server *"chatty"* deve essere eseguito tramite un commando shell della forma *"./<PathToServer>/chatty –f <PathToConfigFIle>"*, dove *"<PathToServer>"* sta ad indicare il percorso verso il file eseguibile del server, mentre *"<PathToConfigFile>"* indica il percorso verso il file di configurazione del server.

Quando il server viene lanciato, dopo aver allocato alcune variabili utili, analizza il file di configurazione che gli viene passato ed estrapola i valori delle variabili per settare il server nel caso in cui questi valori siano presenti, altrimenti utilizza i dati di default preimpostati per ovviare alla mancanza dei primi. Dopo di che può iniziare a configurare il server per ricevere connessioni da client e le conseguenti richieste. Per stabilire le connessioni, il server usa il protocollo TCP gestendo socket AF_UNIX. Per poter riceve connessioni, quindi, apre un socket *listener* che rimane in ascolto per tutta la durata di esecuzione del server. Tutte le connessioni in ingresso stabilite, vengono monitorate dalla funzione *"poll()",* che verifica in ogni istante se un client ha scritto nel socket personale. Quando ciò avviene, la funzione *"poll()"* imposta nella sua struttura dati, *"pollfd"*, una variabile in modo da notificare questo evento. Una volta riconosciuto il socket che ha richiesto l'evento, sarà inserito nella coda delle richieste da gestire e inseguito analizzato da un thread, che si trova all'interno del *pool di thread*. Tale thread esamina il socket che ha all'interno un messaggio (buffer) di tipo *"message_t"* nel cui header è contenuto un valore intero positivo indicante il codice della richiesta inviata dal client da elaborare. Ad ogni richiesta gestita, il server invia al client un messaggio di riscontro esplicitante l'esito dell'operazione, esso è costituito dal solo header del messaggio contente il codice assegnato all'esito.

La chiusura del server avviene tramite l'invio di un segnale di terminazione come: *SIGQUIT*, *SIGTERM* e *SIGINT*. Il segnale ricevuto è gestito utilizzando una variabile globale, impostata dal gestore del segnale, in modo che ogni risorsa allocata possa essere deallocata per rimuovere i dati residui.

Come richiesto nella documentazione ricevuta, è stato gestito anche il segnale USR1, in modo tale che ad ogni suo invio, venga stampato sul file *"chatty_stats.txt",* il resoconto dell'esecuzione del server fino a quel momento.

2. Descrizione

2.1 Gestione delle *connessioni*

La gestione delle connessioni avviene all'interno del file *"chatty.c"*. Essa è effettuata dalla funzione *"poll()"*, che scansiona periodicamente i socket salvati all'interno della sua struttura dati. La struttura dati è composta da tre variabili, ad una viene assegnato il valore del descrittore del file del socket, ad un'altra il codice che stabilisce l'evento, di lettura o scrittura, verificatosi nel socket, alla terza variabile è assegnato il codice di ritorno dell'evento. La funzione "poll()" è preferita alla funzione *"select*()", studiata durante il corso, in quanto permette una migliore gestione dei descrittori dei file. La chiamata di sistema *"poll()"* è disponibile solo per i kernel linux non inferiori alla versione 2.1.23.

Tutti i descrittori dei file sono pre-impostati al valore *"-1"*, quando arriva una richiesta di connessione, viene accettata solo nel caso in cui il numero delle connessioni sia inferiore al valore configurato, quindi, il valore del primo descrittore del file disponibile varia assumendo il valore del descrittore del socket aperto. Se, invece, il descrittore del file è in uso, cioè il socket è impegnato in lettura o scrittura, allora il suo valore viene impostato a *"-2"*, in modo da segnalare al server l'impossibilità di riassegnazione di quel descrittore.

Quando un client ha occupato il socket, e la *"poll()"* lo segnala, il thread principale mette il descrittore del socket in coda ai *tasks* dei *thread workers*, per essere gestito in seguito.

Se durante la scansione dei descrittori dei socket, la "poll()" trova un descrittore privo di evento, esso viene trascurato dall'analisi.

2.2 Gestione delle *richieste*

La gestione delle richieste avviene tramite funzioni implementate all'interno dei file *"chatty.c"* e *"pool.c"*.

La gestione della richiesta in coda comincia quando un *thread worker* acquisisce un socket. Il thread worker è un thread all'interno di una struttura dati denominata *"pool"*, la quale contiene un insieme di thread workers, il numero di thread al lavoro stabilito nel momento della configurazione del server, la coda dei tasks e il contatore dei tasks. Il task è una funzione che deve essere eseguita da un thread worker, in questo specifico caso è la gestione di una richiesta. Obbligatoriamente, gli argomenti di tale funzione devono essere contenuti tutti in

una stessa struttura dati. Il compito del thread workers è quindi quello di prelevare un task dalla coda e mandarlo in esecuzione. Il prelievo del task dalla coda avviene tramite politica FIFO. Se all'interno della coda non vi sono task, i thread workers sono messi in attesa dalla variabile di condizione di quella struttura dati.

La richiesta è gestita in base al codice dell'operazione da eseguire che è nell'header del messaggio inviato dal client. Tutte le richieste, tranne quella di registrazione, devono essere inviate da utenti loggati. L'utente, in questo contesto, è una struttura dati contenente il proprio nickname, il descrittore del file del socket, un flag per determinare se il client ha già ricevuto la cronologia dei messaggi, la coda alla cronologia dei messaggi e una lista con i gruppi nei quali è iscritto.

Le richieste che possono essere gestite dall'utente sono:

- **Registrazione**: essa avviene inserendo il nickname dell'utente all'interno della tabella hash predisposta come elenco utenti e progettata come tabella hash con code di trabocco. Il primo utente corrispondete ad un valore hash, viene inserito direttamente nella tabella, i successivi sono inseriti nella coda di trabocco corrispondente a quel valore hash e a tutti gli utenti viene salvato il descrittore del proprio socket, che varia ad ogni connessione stabilita. Il valore hash viene calcolato usando il modulo tra il numero esadecimale del primo carattere del nickname e il numero di celle della tabella. La tabella hash è stata utilizzata per velocizzare le principali operazioni di ricerca ed inserimento. Finita la registrazione l'utente è inserito nella coda degli utenti attivi. Essa è stata progetta come coda doppiamente concatenata per facilitare l'inserimento e la cancellazione degli utenti attivi.
- **Connessione**: è la ricerca di un utente all'interno della tabella hash, se con esito positivo, esso viene evidenziato restituendo alla funzione le sue strutture dati e reso operativo resettando il valore del descrittore del socket personale ed inserendolo nella coda degli utenti attivi. In caso di esito negativo, invece, verrà chiusa la connessione.
- **Invio della lista utenti**: l'utente che richiede tale operazione riceverà come primo elemento il numero degli utenti attivi e di seguito un buffer con i nickname attivi salvati nella coda FIFO doppiamente concatenata.
- **Disconnessione**: l'utente che la richiede, viene disconnesso dal server, nel contempo il suo nickname viene eliminato dalla coda degli utenti attivi e si azzera anche il descrittore del suo socket.

- **De-registrazione**: l'utente che richiede di deregistrarsi, è rimosso dalla tabella hash, tutte le sue strutture dati e i dati in esse contenuti vengono rimossi. Se vi sono utenti nella coda ti trabocco, il primo viene spostato nella tabella.
- **Creazione di un gruppo**: l'utente che la richiede è impostato nelle informazioni come creatore. I gruppi sono inseriti all'interno di una tabella hash, distinta da quella utenti, ma gestita similmente. Un gruppo è una struttura dati che contiene la denominazione del gruppo, il suo creatore, una coda FIFO con la lista dei partecipanti e una coda FIFO con la cronologia della chat di gruppo.
- **Aggiunta ad un gruppo**: l'utente che ne fa richiesta è aggiunto alla lista dei partecipanti di quel gruppo.
- **Eliminazione di un gruppo o di un utente da un gruppo**: l'utente che richiede questa operazione, se è il creatore, cancella l'intero gruppo, cioè lo elimina dalla tabella hash con tutte le sue strutture dati e di conseguenza tutti i suoi partecipanti. Se invece, l'utente, non è il creatore, questo viene eliminato dalla coda dei partecipanti, mentre il gruppo rimane attivo ed all'interno della tabella.
- **Invio di un messaggio**: l'utente che ne fa richiesta invia un messaggio testuale di tipo *"message_t"* ad un altro utente registrato o ad un gruppo, se il messaggio rispetta la dimensione pre-configurata. Il mittente invia il messaggio al server il quale lo salva nella cronologia dei messaggi del destinatario, se questo è attivo il messaggio viene anche inviato immediatamente al destinatario utilizzando il suo socket. Se il messaggio è destinato ad un gruppo, allora, il server lo salva sia nella cronologia dei messaggi del gruppo sia in quella degli utenti partecipanti e a quelli attivi lo invia direttamente tramite socket personale.
- **Invio a tutti gli utenti registrati**: l'utente che ne fa richiesta invia un messaggio testuale al server il quale lo smista a tutti gli utenti registrati al server, salvandolo nella cronologia dei messaggi di ogni utente. Quelli attivi ricevono il messaggio immediatamente in quanto il server glielo invia utilizzando il socket personale. Il messaggio deve rispettare le dimensioni pre-configurate.
- **Invio di un file**: l'utente che ne fa richiesta invia un file qualsiasi ad un utente o ad un gruppo. Il server riceve come primo elemento il nome del file e poi il file, salvato all'interno del corpo di un secondo messaggio, con la sua dimensione, di conseguenza in questa condizione il server è abilitato a ricevere solo questo tipo di dato. Se la

dimensione del file è minore o uguale a quella pre-configurata allora, questo viene salvato nella cartella di lavoro del server. Per avvisare l'utente o i partecipanti del gruppo che il file è disponibile, viene inviato a questi il percorso del file sotto forma di messaggio testuale.

- **Ricezione di cronologia chat**: l'utente che ne fa richiesta riceve dal server la lista di tutti i messaggi salvati nella sua cronologia che ha una lunghezza stabilita dalla configurazione server e tale lista è un buffer. Se è la prima volta che riceve la lista dopo essersi connesso o dopo la ricezione di altri messaggi viene settata la variabile di invio chat.
- **Ricezione cronologia chat di gruppo**: l'utente che ne fa richiesta riceve dal server la lista di tutti i messaggi scambiati dagli utenti di quel gruppo sotto forma di buffer. Tale lista ha lunghezza stabilita dalla configurazione del server.
- **Download del file**: l'utente che ne fa richiesta riceve il file, se questo esiste, tramite socket sotto forma di un buffer.

Tutti i messaggio ed tutti gli avviso di file da scaricare, arrivati quando l'utente è offline, vengono salvati nella propria cronologia senza rispettarne la lunghezza. Appena l'utente fa richiesta di tale cronologia, la lunghezza della cronologia rientra nei parametri stabiliti dalla configurazione cancellando i messaggi più vecchi.

3. Note

Maggiori dettagli tecnici implementativi sono esplicitati nella relazione contenuta nel file *"ChatterBox_Doxygen.pdf"*, redatta utilizzando il tool da voi consigliato, *"DOXYGEN"*.