Relazione Progetto 1 - Sistemi Operativi
Anno accademico 2013/14

Bortoli Gianluca - matricola n. 159993
Dellera Andrea - matricola n. 158365

6 giugno 2014

1 Prefazione

Il progetto che abbiamo scelto di sviluppare consiste principalmente in un client ed un server che comunicano tra loro, scambiandosi dei messaggi attraverso delle FIFO. Server e client sono due (o più) processi nettamente separati e senza alcun antenato in comune; abbiamo perciò scelto di utilizzare delle *FIFO* (dette anche *named pipe*), invece che delle semplici pipe, per non essere vincolati dal legame di parentela tra i processi.
Presentiamo ora le principali funzionalità che abbiamo attribuito alle due entità e le scelte che abbiamo attuato nell'implementarle.

2 Client

Il processo *client* si occupa, come in una generica architettura client-server, di interrogare un altro processo detto server. Il client si limita quindi ad inoltrare richieste al server, il quale risponderà dopo aver eseguito le operazioni.
Abbiamo attribuito al processo client l'incombenza di prendere in input da riga di comando (usando l'apposita funzione getopt) i seguenti parametri:

1. nome del server a cui collegarsi (**-n**)
2. chiave da utilizzare per la de/codifica (**-k**)
3. un flag nel caso in cui si voglia prendere l'input da file, il quale esclude il passaggio del messaggio direttamente dalla linea di comando (**-f**)
4. due flag per indicare se voglio de/codificare (**-d** e **-e** rispettivamente)
5. nome del file dove, eventualmente, scrivere l'output (**-o**)
6. un flag per poter visualizzare i messaggi precedentemente codificati dal server indicato (**-s**)
7. un indice per richiedere la decodifica di un precedente messaggio (**-i**)

Una volta eseguito il parsing dei parametri da linea di comando, generiamo un nome sempre diverso per ogni client che viene eseguito, in modo da non aver conflitti con i nomi delle FIFO su cui poi i processi vanno a leggere/scrivere.
Per risolvere ciò, abbiamo deciso di concatenare al termine client il *pid* corrispondente (es: client12345).
Infine, avviene la chiamata alla funzione *run_client* con i parametri corretti. Parleremo più approfonditamente delle funzioni particolari implementate, quando tratteremo della libreria *function.h* che abbiamo creato.

3 Server

Il processo *server* gestisce le richieste inviate dal client ed offre essenzialmente il servizio di de/codifica di messaggi scelti dal client.
Esso gestisce tutti gli errori durante il parsing dei parametri da linea di comando, durante la creazione/apertura delle FIFO e per l'eventuale lettura del messaggio da un file esterno.

La chiamata alla funzione *run_server* è stata messa all'interno di un ciclo infinito per far sì che il server rimanga in ascolto finché non viene terminato volontariamente dall'utente, oppure avvenga un errore a causa del quale non possa continuare.
Alla fine, viene eliminata la FIFO relativa al server con quel nome specifico per permettere l'eventuale creazione di un altro server con un nome usato in passato.

4 Libreria functions.h

Il codice è stato strutturato creando una libreria *functions.h* che contiene tutte le funzioni principali. Ciò ci permettere di non appesantire inutilmente i sorgenti del *client* e del *server*.
Le funzioni principali sono la *cript* e la *decript*, la *run_server* e la *run_client* (di cui abbiamo accennato prima) ed infine le funzioni dedicate alla gestione della cronologia dei messaggi dei server.

4.1 cript e decript

Queste due funzioni si occupano, rispettivamente, di criptare e decriptare i messaggi inviati dal client al server secondo le specifiche del progetto.

4.2 run_server

Questo metodo si occupa di gestire l'intera parte del server. Apre le FIFO necessarie nelle modalità corrette, alloca i parametri utilizzati.
La *run_server* gestisce a tutti gli effetti l'intera comunicazione dal lato del server, facendo, quando serve, gli opportuni controlli prima di aprire le FIFO e/o file di configurazione.

4.3 run_client

Questo metodo si occupa, al contrario, di gestire tutta la parte delegata al client, a partire dall'apertura delle FIFO, fino alla scrittura nella FIFO per inviare ad un determinato client ciò che ha richiesto, leggendo i parametri necessari dal file di configurazione creato (e poi aggiornato) all'avvio di ogni server.

4.4 write/read_encoded_message

Queste due funzioni sono state scritte per implementare la possibilità di avere una cronologia dei messaggi che un dato server ha criptato, in seguito alla richiesta di un client.
Per farlo, abbiamo deciso che ogni qualvolta arrivi una richiesta di de/codifica al server, esso crei un file dal nome univoco nel quale memorizza i messaggi in ordine di processamento.
Ciò ci permetterà in un secondo momento di poterli referenziare attaverso un indice e, in questo modo, sarà possibile per il client richiederne la de/codifica senza immettere nuovamente il messaggio.

4.5 show_all_messages

Questa funzione si occupa semplicemente di stampare a video tutti i messaggi precedentemente processati, in modo da permettere all'utente eventualmente di sceglierne uno, riferendosi direttamente all'indice corrispondente.

5 Il log dei server

Durante l'esecuzione viene tenuta traccia di tutti i server con i relativi parametri nel file *lista_server.txt*.
Ciò ci permette una migliore e più semplice gestione dei parametri in input per le varie funzioni che li richiedono, ma, nel contempo, di dare all'utente una panoramica migliore dei server attualmente in esecuzione.
Nel caso in cui un utente, con molti server in esecuzione in background, voglia accedere ai loro servizi, può semplicemente leggere il file di log.
Questa scelta è stata dettata sia dalla comodità di avere una cronologia dei server avviati e per permettere all'utente una visione d'insieme migliore.

6 Makefile

Oltre ai target richiesti, abbiamo deciso di aggiungerne altri per rendere il testing più approfondito e separare la creazione degli eseguibili dalla vera e propria installazione dei programmi *codecclient* e *codecserver*.

1. **install**: sposta gli eseguibili compilati precedentemente dal target *bin* nella cartella /usr/bin/, in modo che siano utilizzabili come comandi della shell
2. **uninstall**: rimuove *codecclient* e *codecserver* dalla cartella /usr/bin/
3. **test**: genera i binari e lancia gli eseguibili per una prova generica del servizio, con il testo del messaggio passato da linea di comando
4. **assets**: genera dei possibili file di input (di prova) dentro la cartella /assets/
5. **testAssets**: testa gli assets creati appositamente dal target precedente, prendendo i messaggi in input da file