UNIVERSITÀ DI PISA

Relazione del progetto OOB-signaling per il corso di Sistemi Operativi

Tony Agosta 544090

Introduzione

Il progetto punta mettere in pratica le tecniche di programmazione nel linguaggio C apprese durante il corso di Sistemi operativi.

Si articola in tre parti fondamentali:

- Il Supervisor;
- Il Server;
- Il Client.

Queste tre "entità" sono collegate fra loro mediante canali di comunicazione, in particolare:

supervisor e server comunicano tramite pipe, server e client comunicano tramite socket.

Oltre a questi tre, sono presenti anche altri file importanti per l'esecuzione stessa del progetto e per analizzare i risultati da esso prodotti: Makefile, mylib, test, misura.

Ognuno di essi verrà analizzato singolarmente di seguito.

SUPERVISOR

Nel supervisor è presente, oltre al codice utile per il normale funzionamento, il *signal handler* ovvero, il gestore dei segnali che determina in che modo il Supervisor deve comportarsi quando riceve dei segnali: in particolari, sono stati gestiti i segnali "SIGINT" e "SIGALARM" grazie anche all'utilizzo di una variabile di tipo *sig_atomic_t* in modo da garantire una esecuzione sicura.

Il gestore dei segnali viene settato all'inizio del programma, dopo di che il Supervisor grazie alla funzione "server_run" manda in esecuzione in server e crea la pipe che utilizzerà per comunicare con essi, salvando i vari PID in un array che poi verrà riutilizzato per effettuare la terminazione dei server avviati.

Nel ciclo di “server_run” leggo dalle pipe che collegano il Supervisor con i relativi server e aggiorno la lista in cui salvo quanto letto dalla pipe.

SERVER

Il server all’avvio riceve due argomenti: l’ID del server ed il File Descriptor della pipe che verrà utilizzata per la comunicazione con il Supervisor. Inoltre, nel main ignoro eventuali “SIGINT” per evitare di far terminare in modo errato il server.

Nella funzione “run_server” inizializzo il socket e mi metto in attesa di connessioni da parte dei client. Per ogni connessione avvenuta con successo creo un nuovo thread che riceverà i messaggi dal client connesso, in modo da ricavare la stima e mandarla poi al Supervisor tramite pipe.

Gestione del multithreading: la connessione server-client è realizzata mediante la tecnica del multithreading, infatti con ogni nuova connessione viene creato un nuovo thread. Le join dei test vengono gestite nella funzione “eliminaSocket”.

Calcolo della stima: il server oltre a gestire la creazione dei socket per comunicare si occupa di calcolare la stima da inviare al supervisor, attraverso la funzione “stampamessaggio”.

In questa funzione leggo dal socket il messaggio che mi viene inviato dal client, cioè l’ID a 64bit. Per calcolare la stima per prima cosa traduco l’id del client in Host Bytes Order e salvo il tempo di arrivo del massaggio. Dopodiché calcolo gli intervalli di tempo successivi tra i vari messaggi e salvo la stima da mandare la minima stima che riesco a calcolare. Ovviamente, se ricevo un solo messaggio non sono in grado di effettuare una stima e quindi al Supervisor non mando nulla.

A questo punto, dopo aver calcolato la stima il server manda (tramite pipe) al Supervisor il suo messaggio composto da: ID del client, il numero di server a cui un particolare client è connesso e la stima calcolata.

CLIENT

Il client genera il proprio Id in modo randomico e moltiplicando tale numero randomico per il PID del client avviato in modo da assicurarmi che il suo ID sia univoco anche nel caso più client vengano generati nello stesso istante. Dopo aver avviato il client richiamo la funzione “servercasuali” in cui creo un array di p posizioni e salvo in ordine sparso i server a cui collegarmi in modo da rispettare la richiesta di collegarsi a p server distinti scelti casualmente. Dopodiché genero in modo casuale il secret del client dando come intervallo di valori [1;3000], e converto l’ID del client in un numero a 64bit con una apposita funzione.

Nel ciclo della funzione “run_client” connetto il client con i p server scelti casualmente tramite socket e una volta avviata la connessione mando al server i w messaggi, inviando l’ID del client convertendolo in Network Bytes Order prima di inviarlo al server.

MAKEFILE

Nel makefile si trovano i target *all, test* e *clean:*

- all per la compilazione di Supervisor, Server e Client;
- test per mandare in esecuzione il test richiesto;
- clean per eliminare i file creati una volta terminata l’esecuzione.

MYLIB

Mylib è una libreria utile da me creata dove sono presenti le varie funzioni e strutture dati utilizzati dal Supervisor, dal Server e dal Client.

TEST

Il test manda in esecuzione il Supervisor direzionando il risultato dell’esecuzione dallo stdout verso un file che poi verrà passato come argomento al file “misura”.

Dopo aver mandato in esecuzione il Supervisor, e quindi i relativi server, inizia un ciclo di dieci iterazioni dove mando in esecuzione a coppie venti client facendo trascorrere un secondo tra una coppia e l’altra, direzionando anche in questo caso il risultato dell’esecuzione dallo stdout verso un file anch’esso utilizzato come argomento per il file “misura”.

Dopo aver terminato il primo ciclo, ne inizia un secondo di sei iterazioni con una pausa di dieci secondi tra un’iterazione e l’altra, dove ad ogni iterazione mando un segnale “SIGINT” al Supervisor che gestirà in modo opportuno grazie al *signal handler*.

Dopo il secondo ciclo mando due segnali “SIGINT” consecutivi al Supervisor in modo da terminarne l’esecuzione grazie alla gestione del doppio SIGINT del *signal handler*.

MISURA

Misura server per calcolare le statistiche e i secret che sono stati stimati in modo errati. Ha come argomenti i due file generati dal file “Test” ed utilizza gli array associativi della bash per salvare i secret e le stime.

Per leggere dai due file uso la read che passo come argomento ad un while e nei due array creati salvo in uno il secret stimato dal supervisor per un determinato client e nell’altro il secreto effettivo di ogni client.

Dopodiché sempre grazie ad un ciclo scorro entrambi gli array per fare le statistiche controllando se il secret stimato dal Supervisor per un client coincide con il secret effettivo di quel client.

Alla fine, stampo sullo stdout i risultati ottenuti.