Relazione Progetto Sistemi Operativi

Marco Colavita, matricola 545104

January 2020

1 Indice

2 Introduzione

- Il progetto presentato descriverà un sistema di out-of-banding-signaling. Tale progetto contiene 5 file necessari per la sua esecuzione: client.c , supervisor.c, Makefile, test.sh e misura.sh. La funzionalità di ogni file sarà descritta in seguito. Ogni funzione è commentata all'interno del codice.

3 client.c

- Il file client.c contiene esclusivamente l'implementazione della parte client. All'avvio il client genera in maniera randomica un secret, (facendo ausilio dalla funzione srand() che prende come seme il pid del processo) ed un ID (generato anch'esso in maniera randomica con la funzione rand_uint64()). Verrà creato poi un array di FileDescriptor di p elementi (p passato da terminale). La funzione connection implementata servirà per creare e connettersi alle p socket a disposizione. Ogni socket è chiamata "OOB-server-i" dove i rappresenta l'iesimo elemento di un array di p possibili server. Questo array, w_server sarà riempito con p valori randomici compresi tra

1 e k, che grazie alla funzione *distinti* non si ripeteranno di valore. Usciti dalla funzione *connection*, i client invieranno w messaggi ai p server connessi; ogni messaggio contiene al suo interno l'ID del client trasformato in network byte order(come specificato nel progetto) che sarà inviato a tutti i server a cui il client è collegato. Dopo l'invio di un messaggio il client aspetterà secret millisecondi prima di ripetere l'operazione. Per l'attesa è stata usata la chiamata di sistema *nanosleep*. Una volta inviati tutti i messaggi, ogni socket sarà chiusa con la chiamata della funzione *all_close*.

4 supervisor.c

- Il file *supervisor.c* contiene l'implementazioni sia dei server che del supervisor. L'intera funzione main è eseguita da parte del supervisor. Il supervisor una volta ricevuti i parametri opportuni andrà a creare un array di pipe anonime; queste pipe serviranno per la comunicazione tra servers e supervisor. Successivamente verranno creati e lanciati i k server utilizzando la funzione *create_server*. All'interno di questa funzione viene eseguita la system call $fork()$ dove nel codice destinato ai figlii (servers) verranno create le socket per la comunicazione con i vari client. I *servers* sono processi multi-thread che si mettono in ascolto sulle socket di indirizzo $OOB - SERVER - i$, mascherano i segnali destinati al supervisor e incaricano il thread *connection* per l'accettazione delle connessioni da parte dei client. Il thread connection entra in un ciclo infinito in cui esegue la chiamata di sistema *accept*, se tale chiamata va a buon fine, il thread connection delega il thread *worker* alla lettura sulle socket e allo scambio di messaggi con il supervisor. Questo thread si mette in attesa sulla socket fino a quando non avrà più da leggere qualcosa, ovvvero quando la socket lato client verrà chiusa. Viene quindi creato un array dove i suoi elementi sono i tempi esatti di ricezione dei messaggi; questi tempi sono presi grazie all'utilizzo della funzione *clock_getttime*. Non ricevendo più messaggi, la socket lato server viene chiusa. Si prende la migliore stima dei tempi con la chiamata della funzione *stima* da me implementata. Quest'ultima mi permette di stimare il miglior intervallo di tempo se e solo se, all'interno dell'array di tempi creato in precedenza vi sono almeno 2 elementi. Trovato questo intervallo, il worker invierà al supervisor, mediante pipe: il messaggio ricevuto, la miglior stima calcolata e quale server ha prestato servizio. Una volta tornati nella funzione main, vengono definiti nuovi gestori per i segnali $SIGINT$ e $SIGALRM$. Infine viene chiamata la funzione $s\_read()$ da me implementata,che, facendo entrare il supervisor in un ciclo infinito, leggerà da ogni pipe tutti i messaggi ricevuti dai servers. In questa funzione viene utilizzata la system call *select* che permette di capire quale file descriptor è pronto. Ogni volta che viene letto un messaggio da parte del server, il supervisor chiamerà la funzione *table* dove si utilizzerà un array di struct per la memorizzazione di id del client della stima e il numero di server che ha fornito la stima. Il

ciclo terminerà quando verrà ricevuto un doppio segnale di SIGINT, dove verrà stampata su *stdout* l'intera tabella costruita. Il segnale SIGINT viene gestito con un contatore globale che, con un segnale di SIGALRM viene resettato dopo un secondo alla ricezione del segnale SIGINT. Se invece il segnale arriva entro un secondo, il supervisor manda un segnale di SIGTERM ai figli, terminando la sua esecuzione.

5 Makefile

- Il file $Makefile$ permette la compilazione dei due file *.c* e l'esecuzione del *test.sh*; in più permette di ripulire la directory di lavoro dai file generati. Il makefile è molto semplice e minimale. il target *all* permette di generare gli eseguibili per *supervisor.c* e per *client.c*. Il target *cleanall* invece ripulisce tutta la directory di lavoro. Infine il target *test* esegue lo script *test.sh*

6 test.sh

- Il file *test.sh* che viene mandato in esecuzione con il comando *maketest* non fa altro che testare tutte le funzionalità del progetto. Viene mandato in esecuzione il supervisor dove vengono ridirezionati i suoi stdout e stderr su due file separti chiamati *supervisor_out* e *supervisor_err*. Successivamente viene assegnata ad una variabile di nome x l'ID del processo. Vengono eseguiti tutti i clients, una coppia ogni secondo, per un totale di 20 clients. Gli output di ognuno vengono ridirezionati su un file di nome *clientout.log*. Nei successivi 60 secondi verranno mandati dei segnali di SIGINT, uno ogni 10 secondi; i thread non terminano l'esecuzione poichè mascherano i segnali destinati al supervisor. Terminato il minuto, il test arresta il processo assegnato alla variabile x con un doppio SIGINT ed immediatamente manderà in esecuzione lo script *misura.sh* che prende come parametri i file *supervisor_out* e *clientout.log* appena creati.

7 misura.sh

- Lo script *misura.h* ricava dai file ricevuti come parametri quanti secret corretti sono stati stimati dal supervisor e a chi appartengono. I secret corretti appartengono al file *clientout.log* mentri quelli stimati al file *superviso_out*. Questi valori sono confrontati tra di loro. Se il valore assoluto della differenza tra i due è minore o uguale a 25 unità di misura, allora il secret stimato dal supervisor è corretto, quindi valido; mentre se è maggiore di 25 allora il secreto non è valido. Lo script stamperà su *stdout* tutti i secret stimati,la loro validità o invalidità, di quanto differescono da quello "vero" e quale client lo aveva generato.