RELAZIONE DEL PROGETTO

Panoramica del progetto:

Il progetto e' composto dai seguenti file:

- client.c
- server.c
- supervisor.c

Questi tre file contengono il codice dei componenti base del progetto. Inoltre sono presenti:

- misura
- testsh

Lo script 'testsh' viene invocato dal target 'test' del makefile, e si occupa di eseguire un test delle varie componenti del progetto. L'output prodotto dal test verrà poi analizzato dallo script 'misura' per ricavare le statistiche finali.

Altri files:

- estab.h/estab.c
- queue.c
- utils.c

I primi due file forniscono un'implementazione della tabella usata dal supervisor per salvare le stime provenienti dai vari server (ho scelto di creare una libreria per nascondere alcune funzioni che non volevo fossero utilizzate dall'esterno), il secondo file è un'implementazione di una coda FIFO dinamica che risulta utile in alcune parti del progetto, mentre l'ultimo file contiene una miscellanea di funzioni, strutture dati e costanti utili.

Di seguito sono riportate le scelte di implementazioni più rilevanti riguardo al progetto:

File client.c:

Ho deciso di inserire come argomento opzionale del 'main' anche il seed di inizializzazione dei numeri pseudo-casuali, in modo che eventualmente possa essere fornito dall'esterno. Questa scelta si è rivelata necessaria quando durante la scrittura di 'testsh', al momento di lanciare i client a coppie, risultava che la coppia di client lanciata insieme aveva lo stesso id: questo era dovuto al fatto che non passava abbastanza tempo tra il lancio di un client e l'altro, quindi la funzione 'time' usata per l'inizializzazione pseudo-casuale ritornava ad entrambi lo stesso valore. È comunque sempre possibile lanciare il client con i soli parametri p k w, in tal caso il client stesso provvederà all'inizializzazione del suo seed.

L'id del client è contenuto in una variabile di tipo 'uint64_t'. Ho optato per questa soluzione dato che il tipo suddetto è standard per ogni macchina in grado di fornire tipi di dato a 64 bit, cosa che vale per la quasi totalità delle macchine attuali. Inoltre questo tipo è garantito che mantenga la dimensione di 8 byte indipendentemente dalle

implementazioni o dal compilatore. Su gcc si sarebbe potuto usare il tipo 'unsigned long long int', ma a differenza di 'uint64_t' non è garantita la portabilità su altri sistemi e/o compilatori (e, comunque, col passare del tempo e l'avanzare della tecnologia l'implementazione di 'unsigned long long int' potrebbe subire variazioni di dimensione anche su gcc).

File server.c:

Dopo aver completato le procedure di inizializzazione, il server entra in un ciclo infinito con il compito di accettare nuove connessioni. Quando un client si connette, il server avvia un thread worker con il compito di leggere i pacchetti e memorizzare il tempo d'arrivo. Il tempo viene salvato sotto forma di millisecondi passati dal 01/01/1970, come da consuetudine UNIX.

Ho preferito usare un server multithread rispetto alla system call 'select' perchè mi sembrava una complicazione inutile gestire il tutto in maniera sincrona, non essendoci tra l'altro strutture dati condivise (l'unica variabile condivisa è 'myself', ma e' sempre acceduta in sola lettura, quindi non ha bisogno di essere acceduta in mutua esclusione). A mio avviso una soluzione multithread, oltre a semplificare la leggibilità del codice, aumenta anche la reattività del server in quanto il dyspatcher, appena creato il worker, può tornare in attesa sulla 'accept' senza dover eseguire altre computazioni.

Per gestire la terminazione del server ho deciso di impostare un handler per il segnale SIGQUIT, che verrà inviato al momento opportuno dal supervisor. In ogni caso l'unica cosa che fa l'handler è rimuovere il socket creato.

Come criterio usato per stimare i secret dei vari client, calcolo il tempo passato tra la ricezione di un messaggio e quello successivo, poi prendo il minimo. In questo modo, se un client invia due messaggi consecutivi allo stesso server ottengo il valore esatto del secret. Inoltre, dovendo scegliere tra due stime, la migliore sarà sempre la minore tra le due. Come prima idea avevo pensato di calcolare l'mcd tra i tempi di interarrivo dei messaggi, ma se qualche errore di misurazione altera anche di poco i valori, questi molto probabilmente risulteranno coprimi tra loro o comunque con un massimo comune divisore anche molto diverso da quello corretto. Inoltre, la soluzione utilizzata funziona sperimentalmente molto bene anche con un numero di client/server elevato.

File supervisor.c:

In contemporanea con la creazione dei processi figli, il supervisor crea un thread per ogni server appena creato, con il compito di ricevere le stime dai server attraverso le pipe anonime e salvarle nella Tabella delle Stime. Dopodiché, il thread principale si mette in attesa del doppio SIGINT in modo sincrono, utilizzando la 'sigwait'. Ho scelto questa soluzione principalmente perchè come risposta alla ricezione del segnale il supervisor deve stampare il contenuto della Tabella delle Stime, utilizzando quindi funzioni di libreria considerate unsafe da usare dentro un signal handler. Inoltre, mentre è in esecuzione l'handler di un segnale, quel segnale viene mascherato

di default, rendendo praticamente impossibile la gestione del doppio SIGINT nel modo specificato dal testo del progetto.

La Tabella delle Stime e' sostanzialmente gestita come un'array dinamico, e le operazioni ad essa associate, oltre all'inizializzazione ed alla distruzione della tabella, sono: l'inserimento di una nuova stima, la lettura di una stima e la stampa delle stime contenute. Tutte queste operazioni sono thread-safe (la mutua esclusione è garantita dal lock interno alla struttura 'estab_t'). L'unica funzione dalla semantica non ovvia è 'add_new_est', che si occupa di inserire nuove stime: se il client a cui è associata la stima da aggiungere non è già presente nella Tabella, 'add_new_est' alloca un elemento in più nella tabella e lo inizializza con i valori della stima da inserire. Se invece il client ha già una stima associata nella Tabella, allora la funzione sostituisce la vecchia stima con la nuova se e solo so la nuova stima è minore (quindi migliore) della vecchia, altrimenti lascia la stima invariata.

File misura:

Prima di tutto, lo script individua quali sono le stime più aggiornate del supervisor usando il comando 'csplit': il file 'supervisor_err' viene diviso in un file per ogni stima che il supervisor ha stampato sul suo stderr (ad ogni SIGINT ricevuto dal supervisor corrisponderà un file). Viene selezionato l'ultimo file, che ha le stime più aggiornate, dopodichè con il comando 'read' per ogni stima del supervisor (quindi per ogni riga del file selezionato) viene cercato il file di output del client con lo stesso id. Se il client viene trovato, si confrontano i secret e si calcolano tutte le statistiche necessarie con 'bc': stime totali, stime correte, percentuale di stime corrette, errore medio per stima, errore medio tenendo conto solo delle stime errate.
Sia 'csplit' che 'bc' sono standard POSIX.

Andrea Valenti