Relazione Out Of Band Signaling

Sistemi Operativi, Università di Pisa.

Martina Trigilia 532155 (corso B)

1. Strutturazione del codice.

Ho ritenuto necessario suddividere su più file il progetto, per avere una migliore leggibilità, gestione e riuso del codice.

1.1 Utility [GESTIONE DEGLI ERRORI]

Questo file contiene la definizione di macro per la gestione degli errori che vengono usate appropriatamente in tutti i rimanenti file. In particolare, abbiamo una macro che controlla se il valore di ritorno di una funzione è -1 e un'altra che controlla se il valore di ritorno di una funzione è NULL e, in tal caso, entrambe settano *errno.*

1.2 Client

Per la generazione del secret e dell'id del client, ho inizializzato il generatore di numeri casuali in modo che gli id dei client fossero univoci anche se lanciati contemporaneamente, facendo uso della struttura timeval e della funzione getpid(). Il client deve scegliere *solo* p dei k server che sono stati creati, e dopodiché si connetterà a questi server. Il client manda *w* messaggi, contenenti il proprio id , ai p server scelti ogni *secret* millisecondi.

1.3 Server

Il server è un processo multi-threaded. Dopo aver creato adeguatamente la socket, il server entra in un ciclo in cui attente connessioni da parte dei client tramite la funzione accept(). Per ogni client che vuole connettersi, viene chiamata la funzione pthread_create che si occupa della creazione di un nuovo thread, a cui sarà associata la funzione di avvio *Comunicazione.*

Nella funzione *Comunicazione* vengono letti i messaggi dei client ,vengono stimati i secret dei client e comunicati al supervisor. Prima che inizi il ciclo in cui vengono letti i messaggi, inizializzo il tempo iniziale, e dentro il ciclo calcolo la differenza tra il tempo in cui è arrivato il messaggio attuale e il

tempo in cui è arrivato l'ultimo messaggio: questo fornirà il tempo di arrivo. Il server sceglie il *minimo* tra i tempi di arrivi e in questo modo calcola la stima del secret del client. Infatti, gli intervalli di tempo che intercorrono tra le ricezioni dei messaggi da parte dei server sono sempre multipli dei secret dei client che li mandano. Quando un client manda per almeno due volte consecutive un messaggio allo stesso server, allora il secret del client viene stimato in modo corretto. Viene usata una struttura dati *msg* per mandare un messaggio che contiene *stima*, *idclient* e *idserver* al supervisor.

1.3 Supervisor

Il supervisor è un processo che tramite la fork genera k processi figli (server) e manda ognuno di essi in esecuzione chiamando la funzione execl. Il supervisor comunica con i server tramite una sola pipe, che viene quindi creata. Il pid di ogni server-figlio viene conservato in un array di k posizioni. Questo array verrà in seguito usato all'uscita dal Supervisor (con atexit registro la funzione apposita) per terminare tutti i figli inviando loro un segnale SIGTERM.
Il supervisor, inoltre, gestisce i segnali SIGINIT e SIGALARM con la funzione sigaction. Mi sono servita del segnale SIGALARM per settare una sveglia da un secondo, in questo modo ho gestito la distinzione tra l'arrivo di segnali SIGINT che sono tra loro intervallati da almeno un secondo, e quelli che sono intervallati da meno di un secondo.
Gli handler di questi due segnali settano la variabile globale *sig_counter* di tipo sig_atomic_t: se *sig_counter* è uguale a 1 allora viene stampata la tabella delle stime su *stdout*, se invece è uguale a 0 la stessa tabella viene stampata su *stderr*.
Il supervisor, nella funzione *comunicazioneServer*, effettua un ciclo in cui attende messaggi sulla pipe da parte dei server. Al supervisor arrivano, per ogni client, *p* stime da *p* server e si serve di una struttura di appoggio per memorizzare queste stime, in modo da poterle stampare quando vengono inviati i segnali SIGINT.
Alla terminazione del supervisor, la funzione *unlink_socket* (registrata tramite atexit) si occupa di cancellare gli indirizzi del dominio della socket relativi ai *k* server.

2. Strutture di appoggio

Il file queue.c contiene l'implementazione di una semplice lista linkata, con le funzioni *add* e *find*. I nodi della lista contengono i campi idclient, stima e num_server (numero di server che ha fornito stime per quel client fino a un dato momento). Per l'inserimento in lista viene semplicemente controllato se

esiste già una stima per quell'id : in caso positivo viene aggiornata la stima solamente se quella ricevuta è più piccola di quella precedentemente salvata, altrimenti viene creato un nuovo nodo per quell'idclient contenente la stima fornita in quel momento.

3. Test

Vengono eseguiti i test specificati nel testo del progetto. Ho fatto sì che gli output in stdout e stderr del Supervisor venissero redirezionati, rispettivamente, nei file di log *supout* e *superr*. Viene fatta anche la redirezione dello stdout del Client in *clientout*.
Il test.sh lancia, infine, lo script *misura*.

4. Makefile

I target richiesti nel testo del progetto sono presenti. Per compilare i file è necessario scrivere da terminare l'istruzione *make* (in questo modo verrà ,inoltre, creata la libreria *libqueue.a*). L'istruzione *make test* deve essere usata per eseguire *test.sh.* Alla fine, *make clean* viene utilizzato per pulire la directory dagli eseguibili e dai file di log.

5. Misura

Con un while vengono letti, tramite redirezionamento dello stdin, i file di log del supervisor e del client. Vengono creati due array associativi, aventi come chiave gli id, che contengono uno le stime e l'altro i secret. Successivamente viene ciclato l'array contenente i secret e vengono calcolate e stampate :

- la percentuale di successi.
- il numero di stime corrette effettuate.
- il numero di stime sbagliate effettuate.
- l'errore medio commesso.