Università di Pisa

Dipartimento di Informatica

Corso di Laurea in Informatica

Progetto di Laboratorio di Sistemi Operativi

Sessione estiva a.a. 2013-14

Relazione sul progetto dello studente Simone Pignotti

Matricola 489911

Giovedì, 19 Giugno 2014 - Pisa

In questa breve relazione vengono descritte maggiormente le scelte progettuali effettuate durante la realizzazione del progetto, e vengono lasciati alla lettura del codice, adeguatamente commentato, i dettagli implementativi meno importanti. Ogni paragrafo consiste della descrizione del contenuto e del funzionamento di ciascuno dei 5 file di cui è composto l'operato.

client.c

Nel file *client.c* è contenuta soltanto l'implementazione del client. Il client è un processo single-threaded che, dopo aver generato la propria coppia *id* - *secret* casualmente (con seed uguale al proprio *pid*, in modo che anche due client lanciati nello stesso istante abbiano diverso *id*), si connette solo ai *p* server distinti che sceglie sempre casualmente, generando un array di indirizzi e uno di socket, entrambi di dimensione *p*, che andranno a contenere rispettivamente gli indirizzi dei server scelti (*OOB-server-i*) e i file descriptor delle socket connesse a tali indirizzi. Fatto ciò, per *w* volte spedisce il messaggio *id* (in *network byte order*) a uno dei server scelti, attendendo poi (*secret**1000) µ*sec* tramite la funzione *usleep*. Infine vengono chiuse tutte le socket, in modo che i server possano procedere con la stima del suo *secret*.

supervisor.c

Nel file *supervisor.c* invece vi è sia l'implementazione del *supervisor* stesso, sia quella dei *server*, che vengono generati dal *supervisor* grazie alla system call *fork()*. Nel *main* del file, che è il codice del ***supervisor***, vengono lanciati i *k* server con i quali si mantiene una comunicazione attraverso *pipe* anonime, delle quali vengono memorizzati i *file descriptor* nell'array globale *pfd*; intanto il *supervisor* tenta di ottenere dalle suddette *pipe* le stime da parte dei *server* con delle *read* non bloccanti, dopo aver instaurato nuove regole per la gestione di *SIGINT* e

SIGALRM. I ***server*** sono processi multi-threaded: si mettono in ascolto sulle socket *AF_UNIX* di indirizzo *"OOB-server-i"*, mascherano il segnale *SIGINT* per non essere interrotti dai segnali destinati al *supervisor*, e delegano l'accettazione delle connessioni al thread ***dispatcher***. Il *dispatcher*, semplicemente, entra in un ciclo infinito in cui per ogni *accept()* correttamente eseguita genera un thread ***worker*** che provvederà alla vera e propria comunicazione. Esso infatti, finchè la socket che gli viene passata come parametro non viene chiusa lato *client*, esegue una *read()* bloccante sulla socket, e quando viene ricevuto un messaggio viene chiamata la funzione *gettimeofday()* per salvare il momento esatto della ricezione del messaggio in un array, da cui verrà poi ricavata la stima del secret come minor differenza tra due elementi consecutivi. Se la stima è consistente, ossia se sono stati ricevuti almeno due messaggi, il *worker* comunica l'*id* del client con il quale ha comunicato (riconvertito in *host byte order*) e la stima del suo *secret* al ***supervisor***, tramite *pfd*. Esso inserisce ogni nuova stima ricevuta in un array di *struct* che ne memorizzano *id* del client a cui si riferisce, stima stessa e numero di server che l'hanno fornita fino a quel momento, mentre se trova già nell'array una stima per lo stesso client la aggiorna, calcolando il massimo comun divisore tra quella trovata nell'array e quella nuova. Questa scelta è stata ben ponderata, e nata dall'osservazione che dopo molte prove su diverse macchine, la stima risultava sempre corretta al millesimo di secondo, se non perchè nessuno dei *p* server avesse ricevuto due messaggi consecutivi. Sicuramente è una scelta rischiosa perchè una minima imprecisione, ad esempio di un millisecondo, comporta un risultato del tutto errato, ma comunque dai test sono emersi risultati molto più precisi rispetto alla precedente scelta di prendere la minima tra le stime dei server, anche grazie al controllo a riga 330: `if (gcd(estim, tab[i].estim)>21)`. Infatti senza questo controllo è molto probabile che due stime inesatte trovino un divisore comune in un numero "piccolo", ed allo stesso tempo è molto poco probabile che venga generato un numero "piccolo" come secret. Così in questo caso il massimo comun divisore è sostituito dalla minima tra le due stime. La gestione del segnale *SIGINT* avviene invece tramite un contatore globale che, tramite un segnale di tipo *SIGALRM*, viene resettato dopo un secondo dalla ricezione del *SIGINT*. Se invece il secondo segnale arriva entro un secondo il *supervisor* manda un segnale *SIGTERM* ai figli (i *server*) e termina esso stesso.

Makefile

Il *Makefile* è molto semplice: il target di default, *all*, compila i file *supervisor.c* e *client.c*, il target *clean* ripulisce la directory, e il target *test* esegue lo script bash *test.sh*.

test.sh

In questo file è definito il comportamento del target *test* del *Makefile.* Anche questo script è molto semplice e non fa altro che lanciare il supervisor e vari client e ridirezionare i loro *stdout* e *stderr* nei file *cliout*, *supserout* e *supsererr*. Ogni 10 secondi, dopo aver finito di lanciare i *clients*, invia il segnale *SIGINT* a `pidof s`, ma esso arriva solo al *supervisor* (anche se i processi server hanno lo stesso nome del supervisor, bisogna ricordare che essi mascherano *SIGINT*), e dopo un minuto ne manda due consecutivi e esegue lo script *misura.sh* sui file creati tramite la ridirezione.

misura.sh

Questo script ricava da *cliout* e *supserout* le informazioni *id* e *secret* tramite pochi passaggi di manipolazione del testo; i *secret* nel primo file sono quelli “veri”, mentre quelli nel secondo sono stimati. Successivamente organizza queste informazioni in un array con lo scopo di confrontarle più agevolmente, e ricava il numero di stime accettabili (ossia con errore di massimo 25 unità), la loro percentuale sul totale di client lanciati e la media aritmetica degli errori sulle stime, e stampa il tutto su *stdout.*