Relazione progetto Laboratorio di Sistemi Operativi: Out-Of-Band Signaling

Alessio Bardelli, 544270

A.A. 2018/2019

Indice

1 Introduzione

Il sistema in oggetto è composto da 3 moduli principali: *client*, *server* e *supervisor*. Sono inoltre presenti 2 strutture dati di appoggio: *dict* e *threadpool*.

L'intero sistema è stato sviluppato in ambiente Unix; implementato tramite il linguaggio C e conforme POSIX. Per la compilazione di ogni modulo è utilizzato lo standard C99.

Ho cercato di utilizzare il più possibile lo stack e di usare le allocazioni sullo heap solo per dati di natura dinamica. Inoltre, la funzione *snprintf()* è stata preferita alla funzione *sprintf()*, in modo da evitare l'overflow dei buffer. Ho anche preferito l'utilizzo di versioni rientranti di funzioni, come la *strtok_r()* e la *strtol()*.

Notiamo in ultimo che nei gestori dei segnali ci si limita ad aggiornare lo stato interno del programma, utilizzando solamente funzioni signal-safe. Questo nell'ottica di evitare la generazione di situazioni inconsistenti come ad esempio se durante l'esecuzione del gestore stesso sopraggiungessero altri segnali. E' gestito esplicitamente il caso di interruzioni indesiderate durante la sospensione della system call *select()*, riattivando la system call interrotta all'interno di un ciclo.

2 Struttura della Root Directory

Nella *root directory* sono presenti i seguenti file:

- **Makefile**, contiene fra gli altri: il target *all* utilizzato per compilare l'intero sistema, il target *test* utilizzato per eseguire un ciclo completo di test sul sistema sviluppato e i target *clean* e *cleanall* utilizzati per ripulire le directory da tutti i vari file generati.
- **Test.sh**, script utilizzato per eseguire un ciclo completo di test, come richiesto dalla specifica del progetto. Se a questo script, viene passato l'argomento - *-debug*, verrà eseguito un test del sistema utilizzando il tool *valgrind* per tener traccia durante l'esecuzione, di eventuali errori o memory leak.
- **Misura.sh**, script utilizzato per analizzare l'output di server, supervisor e client al fine di ottenere delle statistiche su quanti secret sono stati correttamente stimati dai supervisor; come richiesto dalla specifica del progetto.

Sono inoltre presenti le seguenti directory:

- **Header**, contenente tutti i file *.h* necessari alla corretta compilazione del progetto.
- **Src**, contenente i sorgenti relativi ai moduli principali del sistema, quali *server.c client.c* e *supervisor.c*.

- **Lib**, contenente l'implementazione di funzioni e strutture dati di appoggio al sistema. Inoltre in tale directory, saranno generate le librerie statiche relative e, necessarie per la fase di linking.
- **Bin**, directory in cui saranno generati i file eseguibili, relativi ai moduli server, client e supervisor.
- **Log**, directory in cui saranno generati i file contenenti lo standard output ed error dei moduli server, client e supervisor.

E' importante non modificare la struttura della *root directory*, al fine di garantire una corretta compilazione ed esecuzione del sistema in oggetto.

3 Client

Il client per generare il suo secret e id, utilizza la funzione *rand()*, dopo aver adeguatamente impostato il seme iniziale. Tale seme è il risultato della funzione *mix()*, che prende come argomenti il pid del processo client, un'approssimazione del tempo di CPU utilizzata dal programma e il numero di secondi trascorsi dal 01/01/1970 00:00:00.

Per effettuare l'attesa tra l'invio di un messaggio e il successivo è utilizzata la system call *nanosleep()*, come suggerito nella specifica del progetto.

Per garantire una corretta deallocazione della memoria precedentemente allocata, si registra tramite la funzione *atexit()* una funzione di clean up, che si occupa di chiudere tutte le connessioni con i server e di liberare la memoria allocata.

La scelta di un server fra i p a cui il client è collegato, per inviare il messaggio contenente l'id del client, è fatta prima di entrare nel ciclo di invio dei messaggi. Le scelte sono memorizzate in un array di w interi. Questa scelta implementativa, è stata presa nell'ottica di rendere il più efficiente possibile la fase di invio dei messaggi, garantendo così una stima più accurata dei sicret da parte dei server.

4 Server

Il modulo del server ha il thread main che, dopo aver inizializzato tutte le strutture dati necessarie, entrando in un ciclo, fà il dispatch delle richieste di connessione da parte dei client, utilizzando la system call *select()* per controllare che la socket utilizzata per accettare le connessioni, sia pronta.

All'arrivo di una nuova richiesta di connessione da parte di un client, il main thread inserisce un nuovo task nel pool di thread. Il task si occupa di ricevere i messaggi dei client, di fare la stima del sicret e infine, quando il client chiude la connessione, inoltrare la stima appena fatta del sicret al supervisor tramite pipe anonima.

Il server, esce dal ciclo quando il supervisor invia un SIGTERM. Una volta usciti dal ciclo il main thread libera tutta la memoria allocata, chiude tutte le connessioni ed elimina dal file system la socket utilizzata per accettare connessioni.

5 Supervisor

Dopo aver inizializzato tutte le strutture dati e variabili necessarie, dopo aver istallato il gestore dei segnali che, come richiesto dalla specifica del progetto, alla ricezione di un SIGINT stampa su *stderr* le attuali migliori stime dei sicret e con un secondo SIGINT consecutivo stampa su *stdout* le stesse, dopo tutto ciò procede creando i k processi server con pipe anonime associate per la comunicazione interprocesso.

A questo punto il supervisor entra in un ciclo per ricevere le stime dai server e stampare quelle correnti, quando richieste. Alla ricezione di un doppio SIGINT si interrompe il ciclo e si procede chiudendo tutte le pipe anonime, inviando un SIGTERM ai server, liberando tutta la memoria allocata dinamicamente e distruggendo le strutture dati utilizzate.

6 Strutture Dati di Appoggio

6.1 Dict

Tale struttura dati modella un dizionario, ovvero un insieme di coppie chiave-valore. La chiave è un intero univoco di tipo *long long int*. Il valore associato a tale chiave, è un record contenente due interi. Un'istanza di questa struttura dati è mantenuta dal supervisor. Una generica entry *<key, value>* associa all'*id* di un client, la miglior stima del suo *secret* e, il numero di server che hanno contribuito a tale stima. La sua interfaccia è definita nel file *dict.h*, mentre la sua implementazione è contenuta nel file *dict.c*. Tale struttura dati offre le seguenti funzioni:

- **InitDict**, funzione utilizzata per inizializzare correttamente un dizionario.
- **DeleteDict**, funzione utilizzata per distruggere un oggetto di tale tipo.
- **Add**, funzione utilizzata per aggiungere una nuova entry nel dizionario, o per modificare il valore correntemente associato alla chiave se, la chiave è già presente nel dizionario.
- **GetValue**, funzione utilizzata per ottenere il valore associato ad una chiave nel dizionario se, tale associazione esiste; restituisce un valore di default altrimenti.

6.2 Threadpool

Implementa il comportamento di un pool di thread. Ogni processo server inizializza e mantiene un'istanza di questa struttura dati, per gestire tutte le connessione dei client. L'interfaccia e l'implementazione di questa struttura dati sono rispettivamente contenute nei file *threadpool.h* e *threadpool.c*. Tale struttura dati offre le seguenti funzioni:

- **Threadpool_create**, funzione utilizzata per inizializzare correttamente un pool di thread.
- **Threadpool_add**, funzione utilizzata per aggiungere un nuovo task nella coda del pool di thread.
- **Threadpool_destroy**, funzione utilizzata per terminare e distruggere un pool di thread.

Per ulteriori dettagli si consiglia di consultare i file di riferimento (*threadpool.h, threadpool.c*).

7 Compilazione ed Esecuzione

Per la compilazione dell'intero sistema in oggetto, dopo aver scompattato il *.tar* in una directory vuota, spostarsi nella *root directory* del progetto ed eseguire uno tra i due comandi proposti: ***make*** o ***make all***.
Si noti che il target fittizio *all* è il target che viene eseguito di default, come richiesto dalla specifica del progetto. Per eseguire il ciclo completo di test, richiesto dalla specifica del progetto, è sufficiente eseguire sempre dalla *root directory*, il seguente comando: ***make test***.
Tale comando si limita ad eseguire lo script *test.sh*. Tale script lancia un processo supervisor e 20 processi client. Nel lanciare questi processi, si redirige lo *stdout* e *stderr* di ogni processo in file localizzati nella directory *./Log*.

In ultimo, lo script lancia un ulteriore script, chiamato *misura.sh* che, estrarrà dai file prodotti le statistiche sulle stime dei sicret da parte dei server.

Si noti in ultimo che è possibile lanciare lo script *misura.sh* anche manualmente, passando ad esso come argomento, il numero di client che sono stati attivati a patto che, si sia precedentemente rediretto lo *stdout* e *stderr* di processi distinti in file distinti.