Relazione Progetto Laboratorio di Sistemi Operativi

Maria Vitali, 548154

Aprile 2020, Appello Straordinario

1 Introduzione

Il progetto ha lo scopo di implementare un sistema per l'**out-of-band signaling**. Si tratta di una tecnica di comunicazione in cui due entità si scambiano informazioni senza trasmettersele direttamente, ma utilizzando segnalazione collaterale. Lo scopo è quello di rendere difficile l'intercettazione di queste informazioni "private" da parte di un terzo utente che potrebbe star catturando i dati in transito.

In questo specifico progetto, è stato realizzato un sistema client-server in cui i client possiedono un codice segreto (*secret*) che vogliono comunicare a un server centrale, evitando la sua trasmissione diretta. La tecnica di segnalazione collaterale implementata si basa sull'utilizzo di timer (nei client) e su misurazioni dei tempi di ricezione di messaggi consecutivi (nei server).

Un ***client***, infatti, comunica con un server inviando un messaggio. Aspetta poi secret millisecondi prima di inviare il successivo. Un ***server*** misura il tempo che passa tra la ricezione di un messaggio del client e la ricezione del successivo, così da stimare il valore del secret. Le stime raccolte verranno poi comunicate a un server centrale, chiamato ***supervisor***.

Si avranno n client, k server e un supervisor.

I dettagli dei protocolli, delle varie componenti del progetto e delle scelte di implementazione effettuate sono descritti nel seguito.

2 Client

Il client riceve sulla riga di comando tre valori interi: $\boldsymbol{p}$, con $1 \leq p < k$, che corrisponde al numero di server con cui si deve collegare, $\boldsymbol{k}$, numero dei server attivi, e $\boldsymbol{w}$, con $w > 3p$, numero di messaggi da inviare in totale.

All'avvio, genera il suo *secret*, un intero compreso tra 1 e 3000, sfruttando la funzione *rand()*, dopo aver opportunamente fissato un seme sfruttando il *pid* del processo corrente, così da differenziare la selezione di numeri random in client diversi. Genera poi anche un numero casuale a 64 bit, che identificherà quel particolare client in maniera univoca. Per la generazione è stata utilizzata una

combinazione della funzione di libreria *rand()* con l'operazione di AND bit a bit e con degli shift di bit a sinistra.

A questo punto, il client sceglie a caso p server distinti tra i k disponibili, memorizzando i numeri scelti in un array di p posizioni e avvia una connessione tramite socket con ciascuno di loro (utilizzando la funzione *connect(·)* della libreria $< sys/socket.h >$).

Procede successivamente all'invio di w messaggi, distribuiti in maniera casuale tra i server connessi con quel client. Il messaggio consiste in una struttura dati *msg_t*, composta da un intero, che indica il numero di byte del resto delle informazioni attese, e dall'ID unico del client (intero a 64 bit). Il client seleziona quindi in maniera casuale uno dei server con i quali si è connesso, invia il messaggio, dopo aver convertito l'ID in network byte order, scrivendo sulla socket giusta (tramite una write bloccante), aspetta *secret* millisecondi e ricomincia il ciclo scegliendo casualmente un server.

Dopo aver inviato w messaggi, si procede alla chiusura delle socket aperte e alla deallocazione della memoria heap utilizzata.

2.1 Gestione Segnali

Nel client è stato implementato un gestore del segnale SIGPIPE, per permettere una chiusura safe del processo, in caso di interruzione brusca dei server o di problemi nella connessione socket.

3 Server

Il server viene lanciato come processo a sé stante dal supervisor e ha il compito di gestire le connessioni con i clients. Riceverà infatti i messaggi inviati da quest'ultimo e, misurando il tempo trascorso tra la ricezione di due messaggi consecutivi da parte del client i, dovrà stimare il valore di *secret* e comunicarlo al server centrale.

Questo processo riceve da riga di comando due parametri: ***servernum***, numero identificativo unico di quel particolare server, e ***pipess***, numero del descrittore di file della pipe di scrittura per la comunicazione con il supervisor.

3.1 Esecuzione del Server

All'avvio, viene aperta una socket nel dominio AF_UNIX, con nome "OOB-server-***servernum***", e il processo si mette in attesa di connessioni da parte dei clients.

La gestione della connessioni è stata implementata attraverso la System Call *select()*. Questa consente l'esecuzione di un server sequenziale che, attraverso l'uso di set di descrittori di file, permette di capire quali di questi sono "pronti", cioè quali file descriptor richiedono operazioni di accettazione di una connessione (tramite socket), lettura o scrittura di dati. E' stata fatta questa scelta per le proprietà di scaling di questo approccio di gestione di molte connessioni

contemporanee E' stata preferita a un server multithreaded, proprio per evitare continui cambi di contesto e spreco di risorse.

Se il file descriptor della *listening socket* del server è pronto per accettare una nuova connessione (quindi se è arrivata una richiesta da un nuovo client), chiama la SC *accept()* e crea una nuova socket per quella specifica comunicazione. Se invece in uno dei descrittori delle connessioni aperte, presenti nel set della *select()*, è pronta l'operazione di lettura (quindi uno specifico client ha inviato un nuovo messaggio), il server legge l'informazione e aggiorna la stima del relativo *secret*, memorizzandola in una struttura che tiene traccia di tutte le migliori stime per ciascun client collegato.

Nel momento in cui un client con id *id* chiude la connessione con quel server, quest'ultimo invia la sua migliore stima del *secret* corrispondente a *id* al supervisor, scrivendo le informazioni sulla pipe ***pipess***, e toglie il descrittore dal set di file descriptors della *select()*.

3.2 Gestione Segnali

Nel server è stata modificata la gestione dei segnali SIGTERM, SIGINT e SIGPIPE.

SIGINT e SIGPIPE vengono ignorati dal processo, per evitare terminazioni premature dovute a problemi di connessione con singoli client o interruzioni causate da eventi esterni. Si desidera infatti che la chiusura dei server sia esplicitamente guidata dall'attività del server centrale.

Per SIGTERM è stato implementato un handler specifico per permettere una chiusura safe del server, con conseguente deallocazione della memoria heap utilizzata. E' infatti tramite SIGTERM che il supervisor chiude i processi server da lui lanciati, notificando loro la terminazione globale del sistema di OOB-Signaling.

4 Supervisor

Il supervisor è il server centrale del sistema, che raccoglie le stime dei *secret* di tutti i client e calcola quella migliore per ognuno.

All'avvio, riceve da riga di comando un parametro $\boldsymbol{k}$, che indica il numero di server da eseguire. Quindi, lancia tramite la System Call *fork()*, i k processi figli. Per ognuno crea una pipe anonima di comunicazione tale che il supervisor tenga aperto solo il file descriptor del canale di lettura e i server abbiano disponibile solo il descrittore per la scrittura. Inoltre crea un file, *outserver.log*, che sarà destinazione degli stdout e stderr dei nuovi processi. Quest'ultimo è stato aggiunto per evitare sovrapposizioni di stampe e per un eventuale controllo dell'attività dei server.

A questo punto, tramite la SC *execl()*, reinizializza lo spazio di indirizzamento dei processi figli, mandando in esecuzione il codice dei server.

Il supervisor, dopo aver reso la chiamata di sistema *read()* non bloccante, entra in un ciclo di attesa di informazioni provenienti dai server. Controlla in

modo casuale i file descriptors delle pipe aperte e, se sono presenti dati, legge le informazioni e le raccoglie in una lista concatenata, in cui vengono memorizzate le migliori stime relative ai *secret* dei client.

Ricevuto l'ordine di terminare, il processo si chiude come indicato nel paragrafo seguente sulla gestione dei segnali.

4.1 Gestione Segnali

Nel supervisor è stata modificata la gestione dei segnali SIGINT e SIGALRM. E' stato usato un *contatore* per tenere traccia del numero di SIGINT arrivati da quando è stato catturato un segnale ed entro il secondo successivo.

All'arrivo di SIGINT, il *contatore* viene incrementato.
Se quel segnale è il primo SIGINT catturato nell'ultimo secondo ($contatore = 1$), il gestore fa partire un timer (*alarm(1)*).
Quando il timer scatta, quindi nel momento in cui arriva un segnale SIGALRM, se il *contatore* è rimasto uguale a uno, l'handler si occupa di stampare la tabella di informazioni relative alle stime dei *secret* ottenute fino a quel momento e di resettare il *contatore*.
Se, prima della scadenza del timer, viene catturato un secondo segnale SIGINT, quindi il *contatore* risulta uguale a due, il programma termina, stampando la tabella con tutte le informazioni raccolte sui *secret*.

5 Protocolli di Comunicazione

5.1 Client-Server

Il client manda le proprie informazioni al server attraverso una struttura *msg_t*. Invia come primo valore un intero, che specifica la dimensione dei dati che seguiranno subito dopo. In questo particolare progetto, il secondo valore inviato è un intero a 64 bit che corrisponde all'id unico del client. Nel momento in cui il client ha concluso le sue operazioni, comunica la sua terminazione al server inviando una lunghezza del messaggio vuota ($len = 0$), seguita dal suo id *id*. Il server, ricevuto un valore uguale a zero, chiude la socket di collegamento con quel client e invia le informazioni relative al *secret* di *id* al supervisor.

5.2 Server-Supervisor

Il server e il supervisor comunicano attraverso pipe anonime: il primo tiene aperta solo la pipe per la scrittura; il secondo solo quella per la lettura. E' stata utilizzata una struttura dati per la comunicazione di stime, *info*, che permette al supervisor di memorizzare informazioni relative ad ogni client collegatosi ai server disponibili. Il server invia una struttura *info*. Il supervisor la riceve e provvede a fare i suoi calcoli.

6 Librerie

Sono state implementate due librerie di supporto al sistema.

*Nota: Non sono presenti i relativi file **.c** corrispondenti agli header **.h**. Questa scelta è dovuta al fatto che nella libreria < utillib.h > è presente un'unica semplice funzione che ha una vera e propria implementazione, per cui si è deciso di lasciarla nell'header. Il resto del contenuto sono strutture dati e macro. Per quanto riguarda < utilconn.h >, la scelta è stata guidata dall'utilizzo di una libreria simile a quelle proposte dai professori, che non separava l'implementazione delle funzioni presenti. Quindi, seguendo gli esempi visti durante il corso di Laboratorio di Sistemi Operativi, si è deciso di non procedere alla scrittura del codice in file separati e al conseguente linking espicito delle librerie.*

6.1 utillib.h

Contiene elementi di supporto per error checking, comunicazione tra server e supervisor e stima dei secret.

Fornisce macro per il controllo delle System Call e di chiamate a funzioni di librerie standard per l'allocazione della memoria e per la generazione di numeri random.

Definisce inoltre delle strutture dati per la gestione, memorizzazione e comunicazione delle stime del secret dei vari client, facilitando queste operazioni al server e al supervisor.

L'unica funzione presente si occupa della stampa delle informazioni relative ai secret raccolte ed elaborate dal supervisor.

6.2 utilconn.h

Libreria di supporto alla connessione tramite socket tra client e server, basata sulla libreria < *conn.h* >, presente nelle soluzioni agli esercizi proposti durante il corso di Laboratorio di Sistemi Operativi.

Definisce macro e strutture dati per la gestione delle connessioni, parametri necessari per la comunicazione e dimensioni massime dei buffer. Tra queste evidenziamo le macro per la conversione di interi a 64 bit da host a network byte order e viceversa.

Sono implementate inoltre due funzioni per operazioni di lettura e scrittura safe.

7 Compilazione e Test

Per la compilazione e il testing, spostarsi da terminale all'interno della directory con i file del progetto ed eseguire il comando `make`, seguito da uno dei target specificati nel paragrafo 7.1.

7.1 Makefile

Nel Makefile si trovano i seguenti target:

- `all` : genera file eseguibili.
- `test` : dopo aver richiamato il target. `cleanTestFiles`, esegue lo script bash test.sh, per testare il programma. Il richiamare `cleanTestFiles` è utile nel caso in cui si vogliano fare più test consecutivi.
- `cleanTestFiles` : elimina, se presenti, file *.log relativi a test e misurazioni passate. Nel caso in cui questi non siano presenti, stampa un messaggio di errore per segnalarne l'assenza, ma non blocca l'esecuzione degli eventuali comandi successivi nel Makefile.
- `clean` : elimina gli eseguibili
- `cleanall` : elimina gli eseguibili e i file generati dai test (*.log)

Nota: nel caso in cui si esegua il comando `make` *senza target, l'effetto sarà lo stesso di* `make all`.

7.2 Bash Scripts

7.2.1 test.sh

Esegue una betteria di test. Avvia il supervisor che lancia 8 server. Dopo un'attesa di 2 secondi, vengono avviati, a coppie, 20 client, attendendo un secondo tra l'esecuzione di coppie successive.
Ogni 10 secondi viene inviato un SIGINT al supervisor.
Dopo 60 secondi vengono mandati due SIGINT consecutivi. Il supervisor e i server terminano e si manda in esecuzione lo script misura.sh.

Redirezioni di output specificati al momento dell'esecuzione dei processi:
stdout del supervisor - *outsupervisor.log*
stderr del supervisor - *errsupervisor.log*;
stdout dei clients - *outclients.log*.

7.2.2 misura.sh

Esegue delle misurazioni sulla correttezza delle stime dei *secret* dei client da parte del supervisor. Confronta le stime leggendo i file dove sono stati memorizzati gli output dei processi eseguiti. Stampa delle statistiche.