Relazione sul Progetto

Calcolo Distribuito dell'Impronta SHA-256 tramite IPC

Filippo Arduini
VR501479
Corso di Sistemi Operativi 2024/2025

26 giugno 2025

Indice

1 Introduzione

Il progetto ha come obiettivo la realizzazione di un sistema client-server per il calcolo distribuito dell'impronta digitale SHA-256 di un file, tramite meccanismi di comunicazione e sincronizzazione tra processi (IPC) su ambiente Unix/Linux. Il client invia il file al server usando memoria condivisa e segnala la richiesta tramite code di messaggi; il server calcola l'hash e comunica il risultato al client tramite semaforo.

Questa relazione illustra le scelte progettuali adottate, motivandole in dettaglio e confrontandole con alternative possibili.

2 Descrizione del problema

L'esigenza è di calcolare l'hash SHA-256 di un file fornito dal client, utilizzando un server dedicato. La comunicazione tra client e server deve essere efficiente, robusta e sincronizzata, usando IPC standard POSIX/System V.

I requisiti principali sono:

- Trasferire i dati del file dal client al server.
- Segnalare al server la richiesta e la dimensione dei dati.
- Calcolare l'hash e restituirlo al client.
- Evitare race condition e garantire sincronizzazione.

3 Panoramica dell'architettura

Il sistema si compone di due processi:

- **Client**: legge il file, lo scrive in memoria condivisa, invia messaggio e attende il risultato.
- **Server**: riceve richiesta da coda messaggi, calcola hash e scrive risultato in memoria condivisa, segnala client con semaforo.

I meccanismi IPC usati sono:

- **Memoria condivisa** (shmget/shmat) per trasferire dati binari.
- **Coda di messaggi** (msgget/msgrcv/msgsnd) per segnalare richiesta e lunghezza.
- **Semaforo** (semget/semop/semctl) per sincronizzare l'accesso al risultato.

4 Scelte progettuali dettagliate

4.1 Uso della memoria condivisa per il trasferimento del file

Abbiamo scelto la memoria condivisa per il trasferimento del file perché è la modalità IPC più veloce e diretta per grandi quantità di dati. Il client mappa un segmento di memoria condivisa di 1MB e copia il contenuto del file in questa area, che il server legge direttamente.

Motivazioni:

- Minimizza overhead evitando copie multiple.
- Permette accesso diretto e casuale ai dati.
- Facilita la scrittura dell'hash nello stesso buffer.

Dettagli implementativi: La dimensione è fissata a 1MB (definita in `common.h` con `SHM_SIZE`). L'hash SHA-256 (64 caratteri esadecimali + terminatore) è scritto subito dopo i dati del file.

Alternative scartate:

- Pipe/FIFO: inadatte per grandi file e dati binari.
- Code di messaggi: limitate a poche KB per messaggio, inefficienti per file grandi.

4.2 Uso delle code di messaggi per la richiesta

La coda di messaggi serve per notificare al server che è disponibile un nuovo file da elaborare, insieme alla sua dimensione. Abbiamo definito una struttura `msg_request` con tipo e lunghezza del file.

Motivazioni:

- Garantisce un meccanismo affidabile per sincronizzare eventi.
- Il server può bloccare l'attesa senza consumo CPU.
- Separazione netta tra dati (memoria) e segnalazioni (code).

Alternative scartate:

- Polling diretto sulla memoria condivisa → inefficiente.
- Segnali UNIX → più complessi da gestire e meno portabili.

4.3 Uso dei semafori per la sincronizzazione

Il semaforo garantisce che il client non legga l'hash prima che il server lo abbia calcolato e scritto. Il client esegue un decremento (bloccante) sul semaforo, che parte da zero, mentre il server lo incrementa dopo aver terminato.

Motivazioni:

- Sincronizzazione esplicita semplice e affidabile.
- Evita condizioni di gara e accessi non sincronizzati.
- Facile da integrare con meccanismi IPC System V già usati.

Alternative scartate:

- Polling continuo → inefficiente e non elegante.
- Uso di segnali → complessità e rischi di perdita segnale.

4.4 Posizione dell'output dell'hash nella memoria condivisa

L'hash è scritto subito dopo i dati del file nella stessa memoria condivisa, sfruttando la conoscenza della dimensione del file.

Motivazioni:

- Evita la creazione di un'area di memoria separata.
- Semplifica la gestione della memoria e il codice.
- Permette al client di leggere facilmente il risultato.

Alternative scartate: Separare aree per dati e hash avrebbe richiesto più sincronizzazione e gestione.

4.5 Struttura modulare e uso di `common.h`

L'header comune definisce chiavi, dimensioni e strutture condivise tra client e server. Ciò assicura coerenza e facilita modifiche future.

Motivazioni:

- Centralizza definizioni comuni.
- Facilita manutenzione e aggiornamenti.

4.6 Uso di OpenSSL per calcolo hash

La libreria OpenSSL fornisce la funzione `SHA256()`, affidabile e ottimizzata, per calcolare l'impronta.

Motivazioni:

- Evita implementazioni manuali di algoritmi complessi.
- Compatibilità e performance garantite.

4.7 Funzionalità non implementate ma considerate

Sono state valutate funzionalità avanzate (es. gestione concorrente di più client, schedulazione basata sulla dimensione) che però non sono state implementate nella versione base per semplicità e chiarezza.

5 Codice significativo

```
1 fread(shm_ptr, 1, length, fp);
2 fclose(fp);
3
4 struct msg_request req;
5 req.mtype = MSG_TYPE;
6 req.length = length;
7
8 msgsnd(msgid, &req, sizeof(int), 0);
```

```
printf("[CLIENT] Richiesta inviata, in attesa del risultato...\n");

// Attendi il semaforo
struct sembuf sb = {0, -1, 0}; // decrementa
semop(semid, &sb, 1);

// Leggi hash
char hash_output[65];
memcpy(hash_output, shm_ptr + length, 65);
printf("[CLIENT] SHA-256: %s\n", hash_output);
```

Listing 1: Invio richiesta e attesa risultato lato client

```
msgrcv(msgid, &req, sizeof(int), MSG_TYPE, 0);
printf("[SERVER] Ricevuta richiesta di %d byte\n", req.length);

char hash_output[65];
sha256_hash(shm_ptr, req.length, hash_output);

memcpy(shm_ptr + req.length, hash_output, 65);

// Sblocca il semaforo per segnalare il client
struct sembuf sb = {0, 1, 0}; // incrementa
semop(semid, &sb, 1);
```

Listing 2: Ricezione richiesta e calcolo hash lato server

6 Conclusione

Il progetto ha dimostrato l'efficacia della combinazione di memoria condivisa, code di messaggi e semafori per la realizzazione di un sistema IPC robusto e performante. L'uso di librerie standard come OpenSSL ha garantito sicurezza e affidabilità nel calcolo hash. Le scelte progettuali adottate rappresentano un buon compromesso tra efficienza e semplicità di implementazione.

[Filippo Arduini]
Corso di Sistemi Operativi
Anno Accademico 2024/2025