UNIVERSITÀ DEGLI STUDI DI VERONA
DIPARTIMENTO DI INFORMATICA

Corso di Laurea in Informatica

Documentazione del servizio di calcolo dell'impronta SHA-256 di file multipli

Autore: Marco Zordan – Matricola: VR503389
Anno Accademico 2024/2025

Indice

1 Introduzione

Il presente progetto implementa un servizio Client-Server per il calcolo concorrente di impronte crittografiche SHA-256 di file multipli. Il sistema è stato sviluppato attenendosi all'Opzione 2 delle specifiche, che si basa sull'utilizzo delle librerie Pthread e dei meccanismi di comunicazione FIFO (Named Pipes).

Il client ha il compito di inviare le richieste di calcolo, specificando i percorsi dei file. Il server, invece, gestisce in modo efficiente l'elaborazione di queste richieste.

Nel server è stato implementato un thread pool, ovvero, al posto di creare un nuovo thread per ogni richiesta, il server mantiene un numero fisso di thread in attesa.

Per ottimizzare le prestazioni e l'efficienza del sistema si è deciso di integrare diverse funzionalità:

- Caching: Prima di avviare il calcolo di un'impronta, il server controlla se il risultato per il file richiesto è già disponibile in una cache in memoria. Se l'impronta è già stata calcolata di recente, il server restituisce il risultato istantaneamente, eliminando il lavoro ridondante e riducendo drasticamente il tempo di risposta per richieste multiple dello stesso file.
- Scheduling per dimensione del file: Le richieste non vengono elaborate in un semplice ordine FIFO. Il server implementa una logica di scheduling che valuta le dimensioni dei file da elaborare. Questa strategia consente di dare priorità a determinate richieste (ad esempio, quelle per i file più piccoli, per ridurre la latenza media, oppure quelle per i file più grandi, per massimizzare l'uso della CPU), migliorando l'equità e le performance complessive del sistema.
- Gestione ottimizzata delle richieste duplicate: Il server è in grado di identificare richieste identiche per lo stesso file che arrivano in rapida successione. In questi casi, il sistema non avvia un nuovo calcolo. Invece, accoda la richiesta duplicata alla stessa operazione già in corso, garantendo che il risultato, una volta pronto, venga inviato a tutti i client in attesa. Questa funzionalità previene lo spreco di risorse computazionali e mantiene l'efficienza del sistema.

1.1 Esecuzione delle sorgenti

1.1.1 Client

Il compito del client è ricevere in input, da linea di comando, i percorsi dei file per i quali si desidera calcolare l'impronta SHA-256. Una volta che ha ricevuto tutte le risposte il client stampa i risultati a video e termina. Il client segue i seguenti passi:

- Analisi degli argomenti: il client ottiene e controlla gli argomenti forniti in input dall'utente assicurandosi che i percorsi siano validi e accessibili.
- Creazione di una FIFO: per ricevere le risposte in modo univoco il client, crea una propria FIFO, il quale percorso viene generato utilizzando il proprio PID, questo per garantire che il server sappia esattamente a quale client inviare la risposta
- Invio delle richieste: per ogni file inserito da linea di comando, il client formula una richiesta che include sia il percorso del file da calcolare, sia il percorso della sua FIFO di risposta.
- Attesa delle risposte: dopo aver inviato tutte le richieste, il client entra in uno stato di attesa. Apre la sua FIFO di risposta in modalità lettura e si blocca finchè non riceve la risposta

- Visualizzazione risposte: man mano che le risposte arrivano, il client le riceve, le stampa a video e continua l'attesa per la risposta successiva, finché non ha ricevuto tutte le impronte richieste
- Chiusura: una volta che tutte le risposte sono state elaborate il client chiude la sua FIFO di risposta e termina l'esecuzione.

1.1.2 Server

Il server attende le richieste da parte del client fino a quando non viene terminato manualmente. L'eseguibile non richiede argomenti in input. Il server esegue i seguenti passi:

- Creazione della FIFO di richieste: all'avvio il server crea una FIFO di richieste principale, che è l'unico punto di ingresso per tutte le richieste in arrivo dai client.
- Attesa e inserimento in coda: il server si pone in ascolto continuo su questa FIFO, ogni volta che una richiesta da un client viene ricevuta, il server la legge, ne verifica la validità e la inserisce in una coda interna per la successiva elaborazione, questo modo di gestione permette al server di gestire in modo efficiente i flussi di richiesta ad alta velocità.
- Elaborazione e invio delle risposte: per ogni richiesta presente nella coda il server ne prende in carico l'elaborazione:
    - Genera l'impronta SHA-256 o un messaggio di errore in caso in cui il file non esista
    - Utilizza il percorso della FIFO di risposta fornito nella richiesta del client.

Sia client che server possono essere terminati eseguendo il comando "*CTRL+C*" oppure attraverso il segnale "*SIGNINT*".

1.2 Librerie utilizzate

Per l'implementazione delle specifiche sono state utilizzate le seguenti librerie:

- **sha256_file:** Al suo interno, è contenuta una singola e ben definita funzione il cui unico scopo è il calcolo dell'impronta SHA-256 di un file. La funzione prende in input il percorso del file e un buffer fornito dal chiamante, dove viene scritto il risultato dell'hash. La sua logica è completamente isolata dalla gestione delle richieste o dalla comunicazione, incarnando il principio della singola responsabilità.
- **comunication:** gestisce l'intero flusso delle informazioni tra il client e il server. Essa definisce le strutture che incapsulano le richieste e le risposte, standardizzando il formato dei messaggi che viaggiano attraverso le FIFO. Inoltre, la libreria contiene l'implementazione di una coda di richieste, realizzata come una linked list. Questa scelta garantisce una gestione dinamica e efficiente delle richieste in arrivo al server, consentendo l'inserimento e l'estrazione di elementi con un overhead minimo, indipendentemente dal volume di richieste in coda.
- **cache:** Per ottimizzare le prestazioni ed evitare calcoli ridondanti, è stata sviluppata una libreria dedicata alla gestione della cache. La cache è implementata come un array circolare di strutture che memorizzano il percorso del file e il suo hash calcolato. Quando un client richiede l'hash di un file già presente nella cache, il sistema può rispondere quasi istantaneamente, riducendo drasticamente il carico di lavoro del server.
- **pending list:** definisce una lista di richieste che sono attualmente in fase di elaborazione. La pending list consente al server di identificare e gestire le richieste duplicate per lo stesso file:

se una nuova richiesta arriva mentre un calcolo è già in corso, la richiesta viene accodata alla lista di quelle in attesa, anziché avviare un nuovo e inutile calcolo. Questo meccanismo previene lo spreco di risorse computazionali, rendendo il sistema più robusto e performante.

2 Implementazione delle specifiche

- **Comunicazione Client-Server tramite FIFO:** Il server gestisce il flusso di lavoro attraverso la gestione di una FIFO di richiesta principale, che funge da unico punto di ingresso per tutte le richieste dei client. Parallelamente, ogni client crea una FIFO di risposta univoca, il cui percorso viene comunicato al server. Questo meccanismo di comunicazione asincrona e dedicata garantisce che ogni client riceva la propria risposta in modo diretto e sicuro.
- **Pool di Thread Fisso per la Concorrenza:** Per gestire richieste multiple simultaneamente, il server implementa un pool di thread di dimensione fissa. I thread sono "pre-creati" e rimangono sempre attivi, prelevando nuove richieste dalla coda non appena completano l'elaborazione corrente. Questo approccio elimina il sovraccarico di sistema derivante dalla continua creazione e distruzione di thread, garantendo un'alta reattività anche sotto carichi di lavoro intensi.
- **Scheduling per Dimensione del File:** Le richieste in arrivo vengono ordinate nella coda in base alla dimensione del file, con una logica di scheduling che dà priorità ai file più piccoli.
- **Cache in Memoria:** Per evitare calcoli ridondanti, è stato implementato un sistema di cache in memoria. La cache memorizza le coppie percorso-hash dei file già processati. Questo sistema, progettato per essere thread-safe, garantisce un accesso veloce e sicuro alle risposte già calcolate, riducendo drasticamente il tempo di risposta per richieste duplicate.
- **Gestione delle Richieste Duplicate:** Per prevenire lo spreco di risorse, il sistema utilizza una pending_list che tiene traccia dei file attualmente in fase di elaborazione. Quando più client richiedono lo stesso file contemporaneamente, la pending_list permette di identificare la richiesta duplicata ed evitare di avviare un nuovo calcolo. La nuova richiesta viene invece accodata a quella già in corso, e il risultato, una volta pronto, viene inviato a tutti i client in attesa.

3 Strutture dati

3.1 Coda di richieste (queue_t)

```
typedef struct {
    node_t *head;
    pthread_mutex_t mtx;
    pthread_cond_t cond;
} queue_t;
```

La coda mantiene le richieste ordinate per dimensione file crescente

3.2 Sistema di cache (cache_t)

```
typedef struct {
    cache_entry entries[CACHE_SIZE];
    int count;
    pthread_mutex_t mtx;
} cache_t;
```

3.3 Lista Pending (pending_list_t)

```
typedef struct pending_list {
    pthread_mutex_t mtx;
    pthread_cond_t cond;
    struct pending_node *head;
} pending_list_t;
```

4 Difficoltà riscontrate

Durante l'implementazione delle specifiche la difficoltà maggiore è stata la terminazione "pulita" del server. Poiché chiudere il server in modo corretto richiedeva più che fermare il processo principale: era necessario assicurarsi che anche tutti i thread si chiudessero senza blocchi. Il problema era che alcuni thread potevano essere in attesa su una FIFO o su una variabile di condizione. Per gestire questa situazione, è stato adottato un metodo in tre passaggi:

1. Flag di terminazione: quando arriva il segnale di chiusura, si imposta una variabile condivisa che indica a tutti i thread che è ora di terminare.
2. Sblocco dei thread: si invia un segnale (broadcast) che "sveglia" tutti i thread eventualmente in attesa.
3. Chiusura ordinata: i thread, una volta svegli, controllano il flag, completano il lavoro in corso e si chiudono in modo pulito.

In questo modo si evitano blocchi e si garantisce che tutte le risorse vengano liberate correttamente, anche in caso di arresto improvviso.