SHA256 multi-threaded client-server system

UniVR - Dipartimento di Informatica

A.A. 2024/25

Mattia Nicolis – VR500356

Indice

1

Introduzione

Questo progetto implementa un sistema client-server per il calcolo remoto dell'hash SHA256 di un file.
L'architettura è basata su Named Pipes (FIFO) per la comunicazione inter-processo (IPC) su sistemi UNIX (Linux/macOS) e utilizza il Multithreading per gestire richieste concorrenti in modo efficiente.

Il sistema include anche un meccanismo di caching per evitare di ricalcolare l'hash di file già processati e uno scheduler SJF (Shortest Job First) per dare priorità ai file più piccoli.

2

Architettura del sistema

Il sistema è composto da tre componenti principali:

- **Server (server.c)**:
  - gestisce un thread pool (con numero fisso di worker)
  - riceve richieste tramite una FIFO pubblica (/tmp/sha256_server_fifo)
  - Mantiene una Coda di Richieste ordinata per dimensione del file (SJF)
  - Mantiene una Cache (Lista Linkata) degli hash calcolati per ottimizzare le prestazioni
  - Restituisce i risultati tramite FIFO private dedicate a ogni client
- **Client (client.c)**:
  - Prepara una richiesta contenente il percorso del file
  - Crea una FIFO privata univoca basata sul proprio PID
  - Invia la richiesta al server e si mette in attesa bloccante della risposta
- **Monitor cache (cache_query.c)**:
  - Un client speciale che richiede al server lo stato attuale della cache (lista dei file processati e relativi hash)

3

Struttura del codice

File	Descrizone
src/server.c	Il cuore del sistema. Inizializza le FIFO, gestisce i thread, la coda prioritaria e la cache.

4

Dettagli implementativi

Comunicazione (IPC)

La comunicazione avviene tramite FIFO (Named Pipes).

- Canale di Richiesta: Il server ascolta su una FIFO nota (/tmp/sha256_server_fifo). Per garantire compatibilità tra macOS e Linux ed evitare loop di EOF, il server apre la FIFO in modalità O_RDWR.
- Canale di Risposta: Ogni client crea una FIFO temporanea (es. /tmp/client_1234_fifo). Il server apre questa FIFO solo per il tempo necessario a scrivere la risposta.

Gestione della Concorrenza

- Mutex (pthread_mutex_t): Utilizzati per proteggere l'accesso alle risorse condivise: la coda delle richieste e la lista della cache.
- Condition Variables (pthread_cond_t): Utilizzate per sincronizzare i worker thread. I worker dormono finché il main thread non segnala l'arrivo di una nuova richiesta.

Scheduling e Caching

- SJF (Shortest Job First): Le richieste vengono inserite nella coda in ordine crescente in base alla dimensione del file (file_size). I file piccoli vengono elaborati prima.
- Cache: Prima di calcolare l'hash, il worker controlla se il percorso del file è già presente in memoria. In caso di Cache Hit, il calcolo viene saltato e il risultato restituito immediatamente.

5

Compilazione e avvio

Prerequisiti

- Compilatore GCC o Clang.
- CMake e Make.
- Librerie OpenSSL (libssl-dev su Linux, openssl su macOS).

Compilazione Eseguire i seguenti comandi dalla root del progetto: mkdir build

cd build

cmake ..

make

Esecuzione

1. Avviare il Server In un terminale dedicato: ./server (Il server rimarrà in ascolto e stamperà i log delle operazioni.)
2. Eseguire un Client In un altro terminale: # Creare un file di test (opzionale) echo "Contenuto di prova" > test.txt
   # Lanciare il client ./client test.txt (Output atteso:
   Richiesta inviata: test.txt (19 byte) [Calcolato] SHA256: a665a45920422f9d417e4867efdc4fb8a04a1f3fff1f

6

Gestione di errori e casi particolari

- File non trovato: Se il client richiede un file inesistente, il server lo rileva tramite access() e restituisce un codice di errore senza interrompersi.
- Pulizia: Alla chiusura (CTRL+C), il server intercetta il segnale SIGINT e rimuove correttamente la FIFO pubblica.