Relazione sul progetto Client-Server con cifratura XOR

Francesco Zompanti
Corso di Sistemi Operativi 2

May 22, 2025

Obiettivo del Progetto

L'obiettivo del progetto è la realizzazione di un'applicazione client-server per la cifratura e trasmissione sicura di dati testuali o binari tramite socket. Il client legge un file di input, cifra il contenuto tramite XOR con una chiave di 64 bit, e lo invia al server, che decifra e salva il contenuto in un file.

Componenti principali

Il progetto è composto da tre moduli principali:

- **Client** (`client.c`, `client.h`): legge il file, esegue la cifratura in parallelo e invia il messaggio al server.
- **Server** (`server.c`, `server.h`): riceve il messaggio, decifra in parallelo e salva il contenuto su file.
- **Utility** (`utils.c`, `utils.h`): gestisce operazioni comuni come gestione dei segnali, allocazione blocchi, logging, parsing input.

Descrizione del protocollo

Il messaggio inviato dal client è composto da:

- la sequenza cifrata $C = C_1 \| C_2 \| \dots \| C_n$
- la lunghezza L del file originale
- la chiave K usata per la cifratura

Ogni blocco da cifrare ha lunghezza 64 bit (8 byte). Se il file non è multiplo di 8 byte, l'ultimo blocco viene completato con padding (`0x00`).

Funzionamento del client

Il client:

1. Riceve in input il nome del file, chiave K, grado di parallelismo p, IP e porta del server.
2. Divide il contenuto del file in blocchi da 64 bit.
3. Cifra ogni blocco con `XOR` rispetto alla chiave K usando fino a p thread.
4. Invia il messaggio completo al server.
5. Attende acknowledgment.

Funzionamento del server

Il server:

1. Si pone in ascolto sulla porta specificata.
2. Gestisce fino a l connessioni concorrenti.
3. Per ogni connessione, riceve il messaggio cifrato.
4. Decifra ogni blocco con `XOR` rispetto a K usando p thread.
5. Ricostruisce il contenuto originale e lo salva in un file il cui nome inizia con prefisso s.
6. Invia acknowledgment e chiude la connessione.

Gestione concorrenza e segnali

- La cifratura/decifratura è gestita in parallelo tramite `pthread`.
- Sono gestiti i segnali (`SIGINT`, `SIGTERM`, etc.) per impedire interruzioni nei thread critici.
- I dati condivisi sono protetti tramite mutex per evitare race condition.

Test

Sono stati usati i file `sample.txt`, `sample1.txt`, `sample2.txt` per verificare la correttezza della cifratura/decifratura. I file risultanti coincidono con l'input originale (verifica con `diff`).

Conclusioni

Il progetto rispetta tutti i requisiti:

- Comunicazione tramite socket.
- Cifratura e decifratura parallela con grado di parallelismo configurabile.
- Corretta gestione dei segnali e delle race condition.
- Conformità a specifiche client-server date nella traccia.