Progetto Farm - Appello 15/03/2023

Tiziano Calvani 579839

March 27, 2023

1 Introduzione

Il progetto farm implementa lo schema di comunicazione tra processi e thread. Farm main genera dei thread Worker che comunicano con un altro processo denominato Collector che si occupa di raccogliere i messaggi, ordinare i risultati e stamparli sul canale standard output.

Il progetto è stato diviso in diversi file che in fase di compilazioni vengono linkati per creare un codice monolitico che si occupa di effettuare le operazioni richieste.

In particolare, abbiamo:

- generic: in questo file possiamo trovare funzioni utili che vengono usate da tutti i processi;
- queue_lib: dove è presente l'implementazione della coda concorrente, quindi la sua dichiarazione e funzioni che possono essere usate per gestirla;
- signal_handler: contiene tutte le funzioni utili per la gestione dei segnali
- master_worker: si occupa della gestione del thread Pool e dell'inizializzazione della coda concorrente;
- worker: contiene le funzioni utili per la gestioni dei thread concorrenti;
- collector: contiene la parte implementativa del processo che farà da server per la parte corrispondente alla connessione;

2 Struttura

2.1 Main

Il main, contenuto in farm_main.c, per prima cosa gestisce attraverso la funzione getopt le opzioni fornite dall'utente, più in particolare memorizza i parametri che poi verranno utilizzati per l'intera esecuzione del programma.

Dopo aver memorizzato i vari parametri opzionali la funzione main inizializza crea la struttura dati sulla quale avverrà di seguito la connessione. Dopo aver creato e inizializzato la socket, il main eseguirà una fork() cosicché creà un processo Collector che si comporterà da Server e un processo Master Worker che si occuperà di creare un thread pool. Il thread Pool quale si occuperà di creare dei Thread Worker che dopo aver stabilito la connessione con il Server come client distinti, scambieranno messaggi con quest'ultimo.

In conclusione, dopo aver creato il processo Collector il Main attenderà che esso finisca per poi rimuovere il file socket farm.sck.

2.2 Master Worker

Il processo Master Worker si occupa della gestione del thread Pool e dell'inizializzazione della coda concorrente.

La coda concorrente è una struttura dati strutturata in questo modo:

```
 typedef struct {
    char **items;
    int front;
    int rear;
    int size;
    int done;
    pthread\textunderscore mutex\textunderscore t q\textunderscore lock;
} \textunderscore queue;
}
```

Per poter accedere alla struttura sarà necessario acquisire precedentemente la lock attraverso la funzione Pthread_mutex _lock. L'accesso alla coda viene eseguito solitamente attraverso le funzioni enqueue e dequeue che si occupano di controllare che la coda rispettivamente sia non piena e non vuota per poi mettere un item all'interno della coda oppure prelevarlo.

Dopo aver inizializzato la coda, Master Worker crea i thread con all'interno la funzione worker che si occupa di prelevare dalla coda un file name attraverso la funzione dequeue, aprire il file, calcolare il risultato, stabilire la connessione con il Server e poi inviare il nome del file con il rispettivo risultato.

La coda verrà riempita attraverso la funzione enqueue che inserisce il nome del file letto nel argv oppure dalla cartella indicata dal flag -d.

Infine, non appena i file dell'input si sono esauriti, il Master Worker setta il flag -¿done della coda a 1, aspetta la terminazione dei thread, manda un segnale di stop al processo Collector e libera la memoria dinamica allocata per contenere la coda concorrente

2.3 Collector

La creazione dal processo Collector viene eseguita dal Main attraverso la fork().

La gestione delle connessioni viene eseguita attraverso i canali, quindi il Collector è un processo single threaded che gestisce le varie connessioni attraverso il meccanismo della select().

Il Collector inserisce tutti i dati, file_name e result in una lista che poi verrà ordinata per risultato crescente attraverso un selection sort.

Il processo Collector termina quando riceve dai clients il messaggio "STOP". Dopo aver ricevuto il messaggio di terminazione il processo Collector stampa la lista a stdout e libera lo spazio utilizzato per allocare dinamicamente la suddetta lista