Progetto FARM Lab II – Corso A – a.a. 21/22

Alessia Anile
Matricola 619554
maggio 2022

Il progetto contiene, oltre a generafile.c e test.sh forniti precedentemente:

- farm.c
- makefile per la compilazione di farm.c e generafile.c e la rimozione dei file generati.
- boundedqueue.c: implementazione di una coda limitata.
- la cartella utils/includes contenente a sua volta
  - boundedqueue.h
  - util.h: contiene funzioni di utilità, tra cui quelle per la gestione del semaforo.

farm.c

Il programma presenta la gestione di due processi.
Prima di effettuare la fork, viene verificata la presenza di un numero sufficiente di parametri e inizializzato un semaforo con nome, con valore iniziale a 0, così che appena il socket verrà creato si esca dalla regione critica.

Il padre (pid != 0) è il processo *MasterWorker*:

- In caso di presenza di parametri opzionali li legge correttamente assegnandoli alle relative variabili, indipendentemente dall'ordine in cui vengono scritti.
- Alloco lo spazio necessario per gli array di thread worker e delle strutture *thARGS* (di tipo *threadArgs*_t) e *fElem* (di tipo *fElem_t*) e passo alla creazione dei thread worker, i quali eseguiranno la funzione worker.
  - *threadArgs_t* rappresenta una struct con ciò che deve essere passato a ogni thread
    - int thid (opzionale): rappresenta l'id del thread worker.
    - BQueue_t *queue: rappresenta la coda limitata utilizzata.
    - struct sockaddr_un sa: rappresenta l'indirizzo AF_UNIX e serve quindi per poter scrivere sul socket (descritto meglio in seguito).
  - *fElem_t* rappresenta invece un generico file, con il suo nome e il risultato ell'elaborazione
    - char *filename
    - long result
- Prima di creare i thread però, il MasterWorker deve attendere – usando il semaforo – la creazione del socket. Attende quindi con una Wait.
- Dopodichè si passa all'inserimento dei file sulla coda:
  - Controllo il valore di checkTerm: se corrisponde a 1 devo interrompere l'inserimento, continuando solo a elaborare i file già presenti sulla coda; se equivale a 0 posso proseguire.
  - Verifico anche la lunghezza del filename: non deve essere maggiore di 255.
  - Verifico che il file sia regolare usando la funzione isRegular (in util.h) e in tal caso lo inserisco sulla coda.
  - Se non sto lavorando sull'ultimo file passato come parametro, attendo t millisecondi con la funzione sleep. Sleep richiede come parametro un tempo in secondi, perciò effettuo la conversione in millisecondi dividendo per 1000.
- Quando ho finito di inserire tutti gli elementi lo notifico ai thread inserendo una stringa di terminazione sulla coda e impostando checkTerm a 1. Attendo poi la terminazione dei thread con thread_join e procedo a terminare il processo Collector inviando la stringa di terminazione al socket.

- I segnali SIGHUP, SIGINT, SIGQUIT, SIGTERM e SIGPIPE vengono gestiti dall'handlerMW che setta checkTerm a 1.

Il figlio è il processo Collector: per prima cosa esegue una Post sul semaforo: il semaforo serve per attendere la creazione del socket. Collector maschera i segnali gestiti da MasterWorker e usa handlerC per gestire SIGPIPE.
A questo punto Collector, che per il socket prende il ruolo di server, è pronto per attendere i risultati inviati dai thread ogni volta che terminano l'elaborazione di un file.

I thread

Ogni thread ha il compito di elaborare un file alla volta. Ogni volta effettua una pop dalla coda e controlla se il filename è equivalente alla stringa di terminazione. Se lo è, può terminare.
Altrimenti procede a calcolare la quantità N di interi lunghi presenti nel file e li legge, inserendoli in un array. Ottiene poi il risultato desiderato invocando la funzione getResult, la quale chiede come parametri l'array contenente i valori e la dimensione dell'array (quindi N).
Tramite il socket invia poi il risultato a Collector, con la connessione precedentemente stabilita.

Il socket

- **Server:** apre un socket e dopo aver effettuato bind e listen comunica che è pronto tramite una Post. Resta poi in attesa di nuove connessioni tramite la funzione accept.
  Una volta stabilita la connessione, legge ciò che è stato inviato dai thread e lo mostra come output.
- **Client:** dopo essersi connesso al server, scrive il risultato passatogli come parametro sul socket.