UNIVERSITÀ DI PISA

Progetto di Laboratorio di Sistemi Operativi

Anna Francesca Montagnoli

578104 – Corso A

Anno Accademico 2022/2023

Docente Massimo Torquati

Scopo del progetto

Il progetto consiste nel realizzare un programma C, denominato *farm*, composto da due processi.

Il primo, chiamato *MasterWorker,* è un processo multi-threaded composto da un thread *Master* e 'n' thread *Worker*.

Il secondo, chiamato *Collector*, viene generato dal processo *MasterWorker*.

Il programma riceve in input una lista di file binari: ogni file viene inviato ad un thread *Worker*, che legge il contenuto ed effettua un calcolo sugli elementi letti. Il processo *Collector* riceve i risultati di ogni calcolo e li stampa in ordine crescente.

Librerie utilizzate

Per l'implementazione del programma ho scelto di utilizzare alcune librerie che sono state fornite a lezione.

- *util.h* Ho utilizzato le macro *SYSCALL_EXIT*, *CHECK_EQ_EXIT* e *CHECK_NEQ_EXIT* per la gestione degli errori nelle chiamate di sistema, le funzioni *readn* e *writen* per evitare letture e scritture parziali ed ho aggiunto a questa libreria la funzione *EndsWithDat*, utilizzata per controllare se i file passati come input al main sono file .dat.

- *threadpool.h* Utilizzata per implementare il threadpool dei Worker.
  In particolare, gli argomenti passati alla funzione *createThreadPool* (*numthreads* e *pending_size*) corrispondono alle opzioni –n (numero dei threads Worker) e –q (lunghezza della coda dei task pendenti) passate al main.

Inoltre, ho aggiunto la libreria:

- *message.h* Indica il tipo dei messaggi che vengono inviati dai Worker al processo Collector. Contengono il nome del file e il corrispondente risultato del calcolo effettuato su di esso.

farm.c

File che contiene il main.
Controlla gli argomenti passati da riga di comando con la funzione:

```
static void checkargs(int argc, char* argv[])
```

Scelte progettuali:

- Opzione –d: il programma accetta solamente un nome di una directory, non di più.
- Opzioni –n, -q, -t: se gli argomenti di queste opzioni non vengono passati, o non sono numeri, il programma attribuisce loro i valori di default.

Effettua la fork, creando il processo figlio *Collector* e il processo padre *MasterWorker*.
Contiene il thread che si occupa della gestione dei segnali.
Si occupa di controllare i file passati come input, se sono regolari li passa direttamente al threadpool dei workers.

MasterThread.c

Contiene la funzione eseguita dal thread *Master*, invocato dal processo *MasterWorker*. Questo thread si occupa (se viene passata l'opzione -d) di esplorare la directory passata come input al main e tutte le sue sottodirectory, controllando se i file contenuti in esse siano regolari. Per ogni file trovato, aggiunge un task al threadpool. Dopodiché aggiunge al threadpool il task che invierà al Collector un messaggio di terminazione*.

Contiene la funzione:

```
void printSignal(int pfd1)
```

utilizzata dal *signal handler* quando viene ricevuto il segnale SIGUSR1.

Contiene la funzione:

```
int exitMessage(threadpool_t *workers, int fileNum, int sig)
```

utilizzata dal *signal handler* quando viene ricevuto un segnale di terminazione, oppure dal thread *Master* per inviare il messaggio di terminazione regolare.

Worker.c

Contiene la funzione eseguita dai *Worker* del threadpool.

- `void worker(void* arg)` — Legge il file il cui path viene passato come argomento, effettua il calcolo sui suoi elementi ed invia il risultato al processo *Collector*.

* In questo caso l'argomento passato al worker, che generalmente indica il path del file, conterrà la stringa "*exit*". Inoltre, viene utilizzata la variabile *fileNum* (passata dal *MasterThread* come argomento al *Worker*) che indica il numero di file che il Collector deve ricevere prima di terminare la sua esecuzione.

Collector.c

Contiene la funzione eseguita dal processo *Collector*. Aspetta di ricevere tutti i risultati dai *Worker*, dopodiché ordina i risultati in modo crescente, utilizzando la funzione *qsort* e li stampa.

Protocollo di comunicazione

Il processo *Collector* e i thread *Worker* comunicano tra loro attraverso una connessione Socket AF_UNIX.

La socket viene creata in *farm.c* e successivamente il suo file descriptor viene passato come argomento al *Collector*, che nella comunicazione si comporta da server.

Al contrario, i thread *Worker* sono i clients, infatti viene creata una connessione per ogni thread del pool.
In *threadpool.c* nella funzione `static void *workerpool_thread(void *threadpool)`, che viene eseguita per ogni thread, una volta creata la socket, il thread cercherà di connettersi al *Collector*. Il file descriptor della socket viene passato come argomento ad ogni task, in questo modo i *Worker* possono inviare al *Collector* i risultati del calcolo.

Il *Collector* accetta le varie connessioni da parte dei *Worker* e legge i messaggi inviati dai *Worker* utilizzando la *Select*.

Segnali

Il processo *MasterWorker* gestisce i segnali SIGHUP, SIGINT, SIGQUIT, SIGTERM e SIGUSR1 attraverso un thread *signalHandler.*

Quando viene ricevuto uno dei primi quattro segnali, il thread assegna 1 alla variabile *sigFlag* in *MasterThread.c.* Questo indica al thread master che deve terminare quello che sta facendo e inviare un messaggio "*exit*" al processo *Collector.* Quest'ultimo uscirà dal ciclo in cui viene chiamata la select, stamperà i risultati ottenuti fino a quel momento e l'intero programma terminerà.

Quando viene ricevuto il segnale SIGUSR1, il *Master* scrive su una pipe, creata precedentemente, il messaggio "*print*". Quando il *Collector* legge questo messaggio invoca immediatamente la funzione *SortAndPrint*, che ordina i risultati ottenuti fino a quel momento e li stampa, senza però terminare.

Entrambi i processi *MasterWorker* e *Collector* ignorano il segnale SIGPIPE.

Protocollo di terminazione

Il processo *MasterWorker* effettua la *join* sul *MasterThread* e chiama *waitpid* sul processo *Collector*. Quando terminano sia il thread master che il *Collector*, viene inviato un segnale SIGINT al processo *MasterWorker* in modo tale che il thread che gestisce i segnali non si blocchi sulla *sigwait* e termini in modo corretto.

Compilazione ed esecuzione (makefile)

Per lanciare l'esecuzione dello script test.sh scrivere sul terminale "*make test*".

Se si vuole compilare solamente il programma farm, senza il test.sh scrivere "*make farm*".

Nella cartella del progetto è anche presente una directory chiamata "*provafiles*", dove sono presenti vari file .dat creati attraverso il programma *generafile.c*, che ho utilizzato per testare ulteriormente il programma.