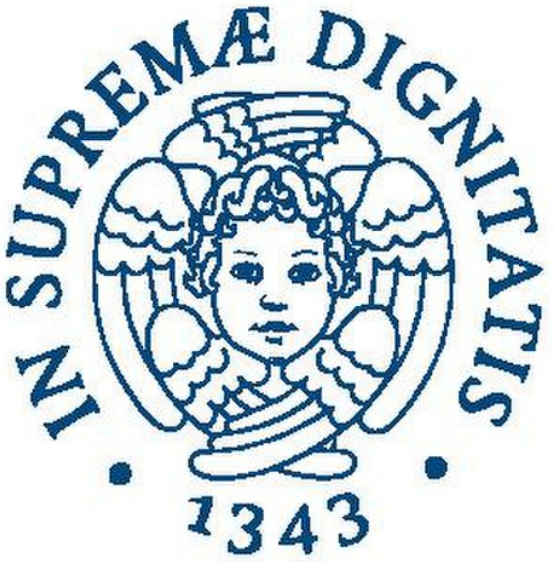

UNIVERSITÀ DI PISA

Progetto di laboratorio di sistemi operativi

A.A:2022/2023

Fabio Bocci
Corso A Matricola 580100

Introduzione

Il progetto si basa sulla creazione di due processi, uno nominato Master Worker, ed uno Collector, i quali comunicano tra loro scambiandosi informazioni sul contenuto dei file letti dal processo Master Worker e al termine o alla ricezione di un segnale il processo Collector stamperà i risultati ottenuti in modo ordinato.
(Documentazione e specifiche richieste dal professore a:
https://github.com/FabioBocci/SOL-2022-23/blob/master/utility/progettoSOLFarm-22_23.pdf)

Implementazione dei processi

Il progetto è strutturato nei seguenti file:

- Collector dove vengono implementate le principali funzionalità del processo Collector
- MasterWorker dove vengono implementate le funzionalità del thread Master che avrà il compito di leggere inserire in una lista condivisa tra tutti i worker i file da elaborare, l'esplorazione delle cartelle, e gestione di interruzioni.
- Worker dove vengono implementate le funzionalità dei thread Worker, lanciati da MasterWorker, rimuovere un file dalla lista, leggere il file, calcolare l'output e mandare il risultato al processo Collector attraverso il socket channel condiviso.
- Main questo è il file che viene usato per lanciare l'esecuzione del programma, dopo aver letto i parametri farà partire il processo MasterWorker e Collector
- Sono presenti anche altri file utilizzati come sostegno al progetto spiegati in (Altri File Utilizzati)

Master Worker

Per inizializzare il processo Master Worker (MW) si dovrà inizialmente creare la nuova struttura dati usando mw_init(..) passando come parametri:

- threadNumber ovvero il numero di thread worker che il processo master worker dovrà generare
- queueLenght ovvero la dimensione della coda dei file che verranno processati contemporaneamente
- directoryName ovvero la directory di partenza dove il processo inizia l'esplorazione
- delay ovvero il delay tra le diverse azioni del processo mw
- pathToSocketFile ovvero il path al socket file dove il processo master worker andrà a comunicare con il processo Collector

una volta creato, sarà possibile avviare il processo mw utilizzando mw_start(..), al quale oltre ad avviare un thread per il processo master worker, è possibile passare una lista dei file da elaborare.
Una volta avviato il processo main potrà aspettare la conclusione usando mw_wait() e pulire la memoria usando mw_destroy();

La funzione principale del processo MW è chiamata master_worker_thread_function(..), e viene avviata usando mw_start().
una volta avviata tenterà la connessione al server (al processo Collector), quindi inizializzerà la threadpool di processi worker con argomenti:

- Queue ovvero la lista sincronizzata in modalità FIFO
- mwFlag ovvero una flag utilizzata dal processo master worker per dire a tutti i processi di smettere di elaborare nuovi dati
- socketChannel ovvero la socket su dove mandare i risultati per comunicare al processo Collector
- masterSocketPrio ovvero una flag usata dal processo master worker per richiedere la priorità sull'utilizzo della socket
- mutex e cond utilizzati per accedere alla socket in modo sequenziale

una volta che il processo ha finito di creare la threadpool inizierà a popolare la lista queue con i file caricati dalla funzione mw_start(), altrimenti se non ci sono altri file da caricare, inizierà ad esplorare le cartelle e sottocartelle partendo da quella selezionata al momento della creazione.
Tra ogni operazione il processo aspetterà un tempo definito al momento della creazione dal valore delay. è anche possibile mandare il ricevimento di un segnale al processo mw usando mw->segnalsHandler, che se uguale a MW_SIGNAL_STAMP_COLLECTOR manderà un segnale al processo collector di stampare tutti i file che ha presenti, invece MW_SIGNAL_STOP_READING si fermerà dal leggere nuovi file o si metterà in attesa che i worker finiscano.
Una volta che il processo non ha più file da caricare nella queue ne cartelle da esplorare, avviserà i worker che una volta che la queue è vuota possono terminare, e una volta attesa che tutti i worker hanno terminato pulirà la memoria occupata da ognuno di essi e terminerà il thread.

Worker

I worker thread della threadpool, creata dal processo master worker, eseguiranno la funzione worker_thread_function(..) dove finchè non riceve il segnale dal processo mw e la queue non è vuota, leggerà un nuovo elemento dalla lista, aprirà il file e calcolerà la somma dei prodotti dei long per l'indice, quindi tenterà di ottenere la lock sull'utilizzo del socket e manderà al processo collector il risultato della somma e il path del file.

Collector

Il processo Collector è un processo mono-thread che una volta avviato, con c_start(), avvierà il server socket ed alla prima connessione stabilità si metterà a leggere i messaggi ricevuti su essa, dove nel caso sia un risultato di un worker lo carica dentro una lista dinamica e ordinata in modo crescente, altrimenti se è un messaggio del processo MW o stamperà i risultati finora ricevuti, o chiuderà la connessione a seconda del tipo di messaggio. Il processo Collector ignora tutti i segnali che non sono stati ricevuti attraverso il socket del processo MW.

Altri file Utilizzati

SynchronizedQueue

Il file SynchronizedQueue gestisce una struttura dati queue sincronizzata di dimensioni fisse al momento della creazione, usando q_init(size) crea una nuova struttura queue al quale si potrà inserire ed estrarre elementi usando q_push() e q_pop(), entrambe queste funzioni restituiscono 0 in caso di successo e -1 in caso di insuccesso, ovvero se c'è stato un timeout sulla wait interna.
L'unica cosa di nota è che durante la funzione pop e push si utlizza internamente pthread_cond_timedwait(...) per mettersi in attesa, ma se capita un timeout questa funzione cercerà comunque di ottenere la lock sul mutex quindi anche in quel caso bisogna liberare il mutex prima di uscire.

DebuggerLevel

Il file DebuggerLevel.h è un file di sole macro che permettono la stampa di messaggi di Log basata su 3 livelli DEBUG_LVL_LOW, DEBUG_LVL_MEDIUM, DEBUG_LVL_HIGH, viene usata per avere messaggi di debug nel codice che possiamo facilmente rimuovere prima del rilascio, definendo M_DEBUG a livello 0 (ovvero disattivato), in questo modo non viene usato un define unico per tutto il progetto ma ogni file che fa l'include del debugger può scegliere a che livello usarlo, così da avere tutte le informazioni che vogliamo senza intasare output del programma.

MemoryManager

Il file MemoryManager.h è un file di sole macro che crea macro per avere un po' più di informazioni sulle allocazioni di memoria e sulle free, utilizzate dal programma, così che prima del termine del programma si possa chiamare la funzione MM_INFO() per sapere se il numero delle malloc e free corrisponde o se abbiamo memory lost di cui dobbiamo occuparci.
Ogni funzione di allocazione e deallocazione oltre a specificare la nuova memoria allocata stampa anche da dove viene allocata usando le macro _FILE_ e _LINE_
Inoltre MemoryManager utilizza DebuggerLevel al livello massimo per le allocazioni di memoria così che se vogliamo rimuovere tutte le stampe di memoria dalla console basta abbassare di 1 il livello di debug.

Scelte progettuali

MasterWorker-worker mono socket channel

I motivi principali per cui ho scelto di implementare la connessione tra i diversi workers, mw al Collector come un solo canale socket sono il fatto che idealmente i worker devono solo mandare un messaggio al collector e poi tornare al lavoro, quindi sarebbe superfluo avere che ogni worker ha il proprio canale socket dove comunicare, dato che rimarrebbe per lo più inutilizzato per la maggior parte del tempo, quindi è più opportuno avere un unico socket condiviso tra tutti i worker.
L'unico problema è che anche il processo master worker ha bisogno di comunicare con il processo collector per inoltrare i segnali, quindi è stato opportuno implementare un sistema di lock con priorità sul canale socket utilizzato per la comunicazione.

Collector mono thread

Avendo la possibilità di dover gestire solo un socket attivo tra MasterWorker e Collector allora è stato possibile utilizzare una struttura mono thread anche per il processo Collector, così da semplificare l'inserimento di dati nella lista ordinata e il ricevimento di segnali dal processo mw.

Repository

link github: https://github.com/FabioBocci/SOL-2022-23
Nel caso il progetto non sia più visualizzabile, poiché privato, contattatemi direttamente a fabiobocci99@gmail.com o su discord a G3m1n1Boy#5243