RELAZIONE: FARM

Progetto SOL - Corso A e B – a.a. 22/23

DEODATI MICHELA 597983

Introduzione:

Il progetto si compone di un totale di 8 file su cui viene separato il codice. I file di maggiore rilevanza sono il “*MasterThread.c*”, “*WorkerPool.c*”, ”*Collector.c*” che impementano sostanzialmete tutte le funzionalità richieste. In generale i file “*WorkerPool.c*” e “*WorkerPool.h*” implementano la threadpool di workers dalla creazione alla distruzione, così come il codice della task che ogni singolo worker deve eseguire. I file “*Collector.c*” e ”*Collector.h*” implementano il processo che stabilisce la connessione con i workers e attende che gli vengano mandati i risultati delle task per poi stamparli. Il processo core di tutto il funzionamento è quello implementato da “*MasterThread.c*” e ”*MasterThread.h*” che svolge il compito di gestore della threadpool e ricettore dei segnali di terminazione e di stampa tramite un thread signal handler. Il controllo dell’input viene lasciato al main Nel file “*Util.h*” sono presenti macro, funzioni e #define che sono di comune utilizzo. La linea generale di funzionamento è: vengono passati file e opzioni di personalizzazione della threadpool come argomenti del main, viene controllato immediatamente che il numero di argomenti sia maggiore di zero,poi viene fatto un controllo più approfondito venfgono esplorate eventuali cartelle, poi viene chiamata funzione ***runMasterThread()*** a cui vengono passati i parametri del main. ***RunMasterThread****()* esegue la fork() per invocare il processo Collector come figlio, crea e avvia il signalhandler e la threadpool, passa i file e la task a ogni worker, attende la terminazione del Collector per terminare anche in caso di ricezione di segnale di terminazione.

WorkerPool:

“WorkerPool.c” e “WorkerPool.h” sono i due file che riguardano la threadpool e la task svolta dai thread. La task consiste nel leggere dei valori **long** maggiori di zero da un file binario passato come stringa alla funzione “*leggieSomma”,* che rappresenta la thread task, calcolare la somma di tutti questi valori moltiplicati per la loro posizione i all’interno del file e comunicare tramite connessione socket AF\_UNIX farm.sck il valore ottenuto e il nome (o percorso) del file al processo Collector. Nello specifico la funzione “***leggieSomma****”* esegue i seguenti passi:  
1. Crea la socket controllando che sia svolto tutto correttamente.  
2. Converte il valore void\*arg nella stringa filePath e apre il file in modalità “rb”(lettura binaria) calcolandone la dimensione tramite il risultato della funzione “***getFileSize()***”, che tramite la fseek() legge la dimensione in byte del file e, sempre tramite la fseek() riportare il puntatore in testa al file e ritorna la dimensione del file.Dividendo poi per sizeof(long) la dimensione in byte ottenuta da questa funzione ottengo quanti numeri ci sono esattamente nel file, a questo punto viene allocato un fileArray che conterrà i valori letti dal file.  
3. Leggo tutti i valori del file con la fread, passandogli direttamente la dimensione del file e il puntatore alla testa del fileArray allocato dinamicamente, eseguo la somma, alloco il buffer per scrivere e aspetto che la connect sia diversa da -1, cioè che il Collector faccia accept().  
4. Scrivo invio filePath e valore del file al Collector assicurandomi che venga scritto ogni byte in una sola volta.  
  
Parliamo ora delle funzioni che gestiscono la threadpool dalla creazione, alla distruzione. Ho definito tre struct: “*workerthread\_t*”, ”*workertask\_t*” e ”*workerpool\_t*” che rappresentano rispettivamente un genrico thread apprtenetente alla pool, la funzione e gli argomenti della taskdi ogni worker, la threadpool con tutti i suoi parametri.  
La funzione **createWorkerpool** crea il toto la threadpool: alloca la coda delle task e inizializza i parametri come il numero massimo di thread, la mutex sulla coda, le condizioni di accesso alla stessa, gestisce gli errori nella creazione della threadpool stampando all’occorrenza cosa è andato storto e ritornando NULL in caso di insuccesso. Associa ad ogni thread l’indice corrispondente nella threadpool. I thread creati non vengono inizializzati direttamente con la task *leggieSomma*  ma vengono inizializzati con la generica task “***wpoolWorker***” a cui viene passata la threadpool, nella quale il thread tenta di acquisire la mutex sulla coda per rimuovere una task ed eseguirla, ma solo se la coda ha task da eseguire e se non c’è un altro thread che cerca di accedere alla coda. Dopo aver acquisito la lock la prima cosa che fa è controllare se la pool non è in fase di uscita, altrimenti non vengono accettate nuove task, e controllare che la coda delle task non si vuota. Se la pool fosse in fase di uscita si controlla se è un’uscita forzata o un’uscita con attesa di terminazione delle task già inserite in coda, nel primo caso il thread ritorna immediatamente, nel secondo caso il thread continua ad iterare, anche se si è in fase di uscita, in modo da finire le task in coda.Quando finiscono le task in coda(pendingQueueCount=0) il thread termina.  
La funzione **destroyWorkerpool** è la funzione che si occupa di liberare la threadpool. Vengono passati come argomenti la pool e un bool che rappresenta il segnale di uscita forzata, dopo aver controllato che i parametri passati siano validi invia segnali tramite la *“pthread\_cond\_signal()”* ai workers fino a che non si svuota la coda. Dopo di che aspetta tramite la pthread\_join su ciascun thread della threadpool che abbiano finito di eseguire le loro task, dopo di che libera la memoria occupata dai parametri della pool con la **freeWorkPool()**  che libera tutte le risorse occupare dalla pool.  
Per gestire l’aggiunta di task alla coda c’è la **addTask**, che prende come parametri, la pool, un puntatore a funzione e l’argomento della funzione che viene passata come puntatore. La prima cosa che fa è controllare che gli argomenti passati siano validi dopo di che inserisce la task in coda ed controlla che il puntatore alla fine della coda non abbia superato la dimensione della stessa altrimenti lo riporta in testa. Dopo di che lancia una signal e rilascia la coda.

MasterThread:

Il processo MasterThread è il core del progetto implementa la gestione dei segnali e la gestione della threadpool. La funzione principale è ***“runMasterThread()”*** che come prima cosa alloca le risorse per la threadpool, gestisce poi il segnale SIGPIPE per evitare di terminare da una scrittura su una socket. Dopo di che crea una pipe senza nome che serve da comunicazione tra il signal handler e il Collector principalmente, ma anche come mezzo di comunicazione tra il MasterThread stesso e il Collector in caso di errore. Esegue la fork() controllando che non ritorni un valore negativo, altrimenti termina, nel processo figlio(pid=0) chiama la funzione che avvia il Collector. Nel ramo “parent” crea il avvia il signal handler in modalità detach in modo che una volta terminato le risorse venganop retituite direttamente al sistema senza che ci sia bisogno di fare la join. Se fino a qui non ci sono stati errori inizia la fase di passaggio delle task una alla volta alla threadpool tramite la funzione **addTask()**

Collector:

Nelle specifiche del progetto era stata lasciata libera la scelta se usare il MasterThread o il Collector a fare da server per la comunicazione dei risultati calcolati dai worker. In questo caso ho implementato il Collector come il server della connessione principalmente per due motivi: in questo modo i workers non devono condividere una singola socket su cui scrivere ma ciascuno ha la propria che gestisce da se dall’apertura alla chiusura. Secondo motivo è la terminazione e la stampa tramite ricezione di segnale: realizzando il collector come un server ho potuto aggiungere al master set la pipe *signal\_pipe* precedentemente allocata nel MasterThread sulla quale il thread signalHandler può scrivere “s” in caso di ricezione di segnale SIGUSR1 per notificare al Collector che deve stampare e “t” in caso di ricezioni di segnali di terminazione per notificare al Collector di terminare l’esecuzione.

main.c:

Util.h:

Informazioni sulla compilazione: