Giovanni Salinitro n°matricola 560226

RELAZIONE PROGETTO

ISTRUZIONI PER LA COMPILAZIONE ED ESECUZIONE:

Per la compilazione del progetto usare il comando “make” in modo che vengano eseguite le regole di compilazione del Makefile; al termine dell’esecuzione di tale comando, dunque, ci saranno due eseguibili, “farm” e “generafile”, necessari per l’esecuzione dello script test.sh.
Per eseguire lo script test.sh basterà, dopo aver lanciato il comando “make” e quindi dopo che gli eseguibili “farm” e “generafile” siano stati creati, digitare il comando “make test” e lo script test.sh verrà lanciato ed eseguito.

SCHEMA GENERALE E BREVE DESCRIZIONE DELLE SCELTE IMPLEMENTATIVE FATTE:

Per ciò che concerne lo schema generale del progetto all’esecuzione del programma, il main, dopo aver creato il file socket “farm.sck” e avergli assegnato un indirizzo, eseguirà una fork per la creazione dei due processi “Master”(che sarà il processo padre) e “Collector”(che sarà il processo figlio) e quindi questi entreranno in esecuzione. Il Master gestirà i segnali tramite un thread dedicato eseguendo la sigwait di tali, poi farà un check sugli argomenti passati da linea di comando per verificare la correttezza dei parametri. All’interno del check, se verrà passata la lista di file binari, si controllerà se siano file regolari e in tal caso i loro path verranno messi in una coda chiamata “coda1”; se invece verrà passata una directory con l’opzione -d, si visiterà la directory e le eventuali sottodirectory, controllando se i file all’interno siano regolari e binari e, nel caso in cui lo siano, i loro relativi path verranno messi sempre in “coda1”. Fatto ciò, il Master eseguirà la accept aspettando che il Collector si colleghi tramite la connessione socket e creerà il thread Master e i vari thread Worker, per poi eseguire la join di questi ultimi e terminare. Il thread Master preleverà gli elementi da “coda1” per inserirli in un’altra coda, “coda2”, che sarà la nostra “coda concorrente dei task da elaborare”, e quindi dove i worker lavoreranno concorrentemente per eseguire il calcolo sui vari file e mandare i risultati con il path relativo dei file, tramite la connessione socket, al Collector. Il thread Master prende come argomenti la coda1, la coda 2, il delay che deve intercorrere tra due successivi inserimenti nella coda concorrente dei task e il file descriptor della socket, mentre i vari Worker avranno come unico argomento la coda2. La coda2 è una coda, con un parametro size per indicarne la dimensione, composta da elementi che hanno come parametri una stringa che sarà il path del file, un long che sarà il risultato del calcolo svolto e un int che indica il file descriptor della socket affinchè i Worker possano mandare i dati al Collector. Il Collector avvierà la funzione per mascherare i segnali, dopo di che si connetterà al Master

tramite sempre la connessione socket e starà in ascolto per ricevere i dati che i Worker gli manderanno e li metterà in una coda chiamata “result”; questa è appunto una coda dove vengono inseriti i vari risultati, in particolar modo è una coda composta da elementi che hanno come parametri un long che indica il risultato del calcolo e una stringa che sarà il path del file. Una volta che i Worker avranno mandato tutti i dati al Collector, questo svolgerà il sorting degli elementi della coda dei risultati e li stamperà a schermo, per poi terminare. Per la gestione del segnale SIGPIPE il Collector lo ignorerà semplicemente in quanto leggerà solamente dalla socket e non ci scriverà mai, mentre il Master farà terminare i vari thread Master e Worker in quanto, ricevendo tale segnale, vuol dire che il Collector non sta piu in ascolto (chiaramente tale gestione viene svolta dal tthread dedicato per la gestione dei vari segnali). Per far in modo che il thread Master termini alla ricezione di determinati segnali per bloccare l’immissione di nuovi elementi nella coda concorrente dei task, ho usato una variabile volatile chiamata “Exit” che è sostanzialmente un flag dove, quando il processo Master riceve il determinato segnale, viene settata a 1 e il thread Master finirà di svolgere il proprio lavoro e terminerà. Quando il thread Master finirà setterà un flag “finito” a 1, in modo che lo sappiano i vari Worker, e risveglierà questi ultimi .Infine, per introdurre il delay richiesto con l’opzione -t ho usato una struct timespec e poi nel thread Master ho chiamato la funzione clock_nanosleep con un CLOCK_MONOTONIC per far scorrere il tempo richiesto, in quanto è una funzione molto affidabile per la gestione di ritardi ben precisi.

DESCRIZIONE FILE HEADER USATI:

I vari file header usati si trovano nella cartella “utils” e sono:

util.h: file header fornito dal Professore usato per fare le chiamate delle varie System Call con gestione di errori

conn.h: file header dal Professore per l’implementazione di readn e writen per evitare scritture e letture parziali

check_argumets.h: qui abbiamo la definizione della funzione check_arguments, la cui implementazione è nel file check_arguments.c

Collector.h: header per la definizione della funzione CollectorExec per l’esecuzione del Collector

Master.h: header per la definizione della funzione MasterExec per l’esecuzione del Master

ConcurrentQueue.h: header per la definizione della coda concorrente usata dai thread, degli elementi (Node) che verranno inseriti e delle varie funzioni associate ad essa

Result.h: header usato per la definizione della coda dei risultati usata dal Collector, dei suoi elementi e delle varie funzioni associata ad essa

Workers.h: header usato per la definizione delle funzioni passate al thread Master e thread Worker e della struct usata per il passaggio degli argomenti al thread Master

DESCRIZIONE FILE.C :

main.c: contiene il main per l'esecuzione del programma
Collector.c : file contenente il codice che implementa il Collector
Master.c : file contenente il codice che implementa il Master e il thread dedicato alla gestione dei segnali
check_arguments.c : file contenente il codice che implementa la funzione che controlla i parametri forniti da linea di comando, due funzioni che verificano se un file è regolare e binario e una funzione "navigateDirectory" per navigare la directory passata da linea di comando
ConcurrentQueue.c : file contenente il codice che implementa le varie funzioni della coda concorrente usata dal thread Master e thread Worker
Result.c : file contente il codice che implementa le varie funzioni della coda dei risultati usata dal Collector
Workers.c : file contente il codice che implementa le funzioni passate al thread Master e ai thread Worker e la funzione per svolgere il calcolo richiesto nei vari file binari.
Infine abbiamo il generafile.c e test.sh forniti dal Professore.