RELAZIONE PROGETTO LABORATORIO SISTEMI OPERATIVI

CECCHETTI CHIARA 583054

SESSIONE DI GENNAIO 2023

Overview:

Il programma farm si articola di due processi, il processo collector, lato server, accetta le richieste di connessione e gestisce i file in arrivo memorizzandoli in ordine non decrescente di risultato, il processo masterthread invece si occupa di elaborare i vari file passati come argomento e inviare i risultati dell'elaborazione al processo collector, che ne stamperà poi l'elenco.

Processo collector:

dopo aver mascherato i vari segnali, compreso SIGPIPE, viene instaurata la connessione lato server e il processo entra in un ciclo dove rimarrà in attesa di ricevere informazioni o sulla pipe condivisa col processo padre o sulla socket, a seconda del tipo, verranno soddisfatte richieste diverse.

pipe -> ricevuto un segnale, devo terminare o stampare la lista dei risultati

socket -> ricevuto una nuova richiesta di connessione o è stato ricevuto un nuovo file da dover memorizzare da parte di un client già connesso.

Il processo termina alla sola ricezione di un segnale di terminazione, disconnettendosi dal socket e liberando la memoria occupata per l'elaborazione.

Funzioni:

- `void stampacoda(Queue_t *q)`: data una coda come parametro, stamperà il contenuto di quest'ultima

- `Queue_t *pushOrdered(Queue_t *coda, msg_t *msg)`: data una coda, e un messaggio, la funzione restituirà la coda con all'interno il messaggio inserito ordinatamente se tutto è andato a buon fine, altrimenti NULL

- `int cmd(int connfd, Queue_t *coda)`: funzione che gestisce una singola richiesta da parte del client, ritorna 0 se la richiesta è giunta a termine in modo corretto, -1 altrimenti.

Processo masterthread:

Immediatamente dopo il mascheramento dei segnali verrà lanciato il thread che si occupa della gestione di quest'ultimi.
Successivamente, si procede al parsing delle opzioni da linea di comando e subito dopo verrà lanciato il thread in veste di master del pool di thread che si occupano dell'elaborazione dei file.
Dopo aver atteso il termine del thread master si procede verso la terminazione, inviando un segnale al signal_handler che ne comporta la sua terminazione e di conseguenza la terminazione del processo figlio (collector).
Dopo aver atteso la suddetta terminazione anche il processo masterthread procederà alla pulizia delle strutture dati allocate durante la sua esecuzione e terminerà.

Funzioni:

- `void usage (const char* argv0)`: funzione di utilità, stampa a schermo il corretto utilizzo del programma.
- `int isdot (const char dir[])`: funzione di utilità, controlla se la directory passata come parametro è la cartella stessa o la cartella padre.
- `void lsR (char* files[], const char *nd,long*n, int* index)`: procedura che mi permette di visitare ricorsivamente la cartella passata come argomento e controllare che tutti i file al suo interno siano regolari per poterli aggiungere alla coda di file da dover elaborare.
- `args_t* checkargs (int argc, char* args[])`: funzione che si occupa dell'intero parsing delle opzioni date da linea di comando, ritorna una struct dove all'interno sono settati i valori desiderati ed è presente l'elenco completo dei file regolari da dover processare.
- `void *sigHandler(void *args)`: task associato al thread che gestisce i vari segnali, alla ricezione di uno di essi, scriverà sulla pipe il nome di tale segnale. Termina solo se ne riceve uno di terminazione. Deve ignorare SIGPIPE.
- `void *master (void *args)`: task associato al thread master, dopo aver bloccato i vari segnali, si occupa di far partire il pool e pushare sulla coda condivisa dai workers i file che devono essere elaborati. Termina alla ricezione di un segnale di terminazione o dopo aver pushato tutti i file passati da linea di comando. Attende la terminazione dei worker nel pool, pulisce le strutture dati utilizzate e termina.

- `void *task (void *args)`: task associato a un generico worker del pool. Dopo aver bloccato i vari segnali, compreso SIGPIPE, tenta di connettersi al server (processo figlio, collector) e successivamente inizierà a prelevare uno alla volta, i nomi dei file da elaborare dalla coda condivisa, ne calcolerà il risultato e, se nel frattempo, non ho ricevuto un segnale, invierà il tutto sulla socket. Il procedimento terminerà alla ricezione di un segnale o al termine dei file da elaborare, chiuderà la connessione lato client e termina.

Gestione segnali:

prima di eseguire la fork, viene dichiarata una pipe senza nome che verrà utilizzata dai due processi per potersi comunicare l'arrivo dei diversi segnali: il thread signal_handler all'interno del processo masterthread, scriverà sulla pipe, mentre il processo collector leggerà dalla pipe il tipo di segnale ricevuto e agirà di conseguenza.

Compilazione ed esecuzione:

Per compilare il programma basterà posizionarsi all'interno della cartella dove è presente il file farm.c e digitare da linea di comando "make all".
Successivamente, per poter eseguire i test basterà digitare, sempre da linea di comando: "bash test.sh" e attendere.
Per poter ripulire la cartella dopo l'esecuzione, digitare da linea di comando: "make cleanall".