

Cordoni Rebecca 580950

Corso A

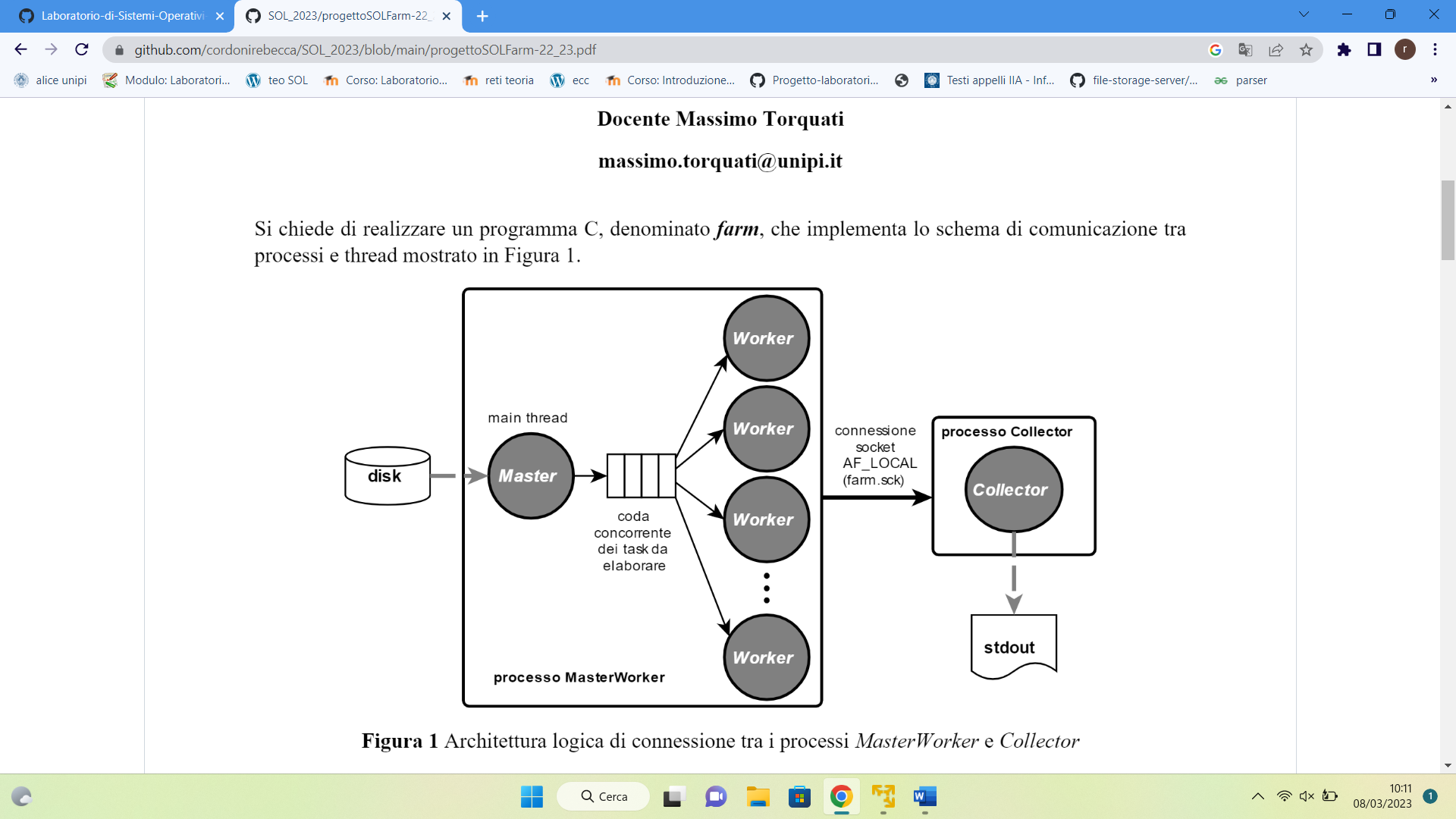
Relazione per il progetto “farm”

Sistemi operativi

2022-2023

**Introduzione**

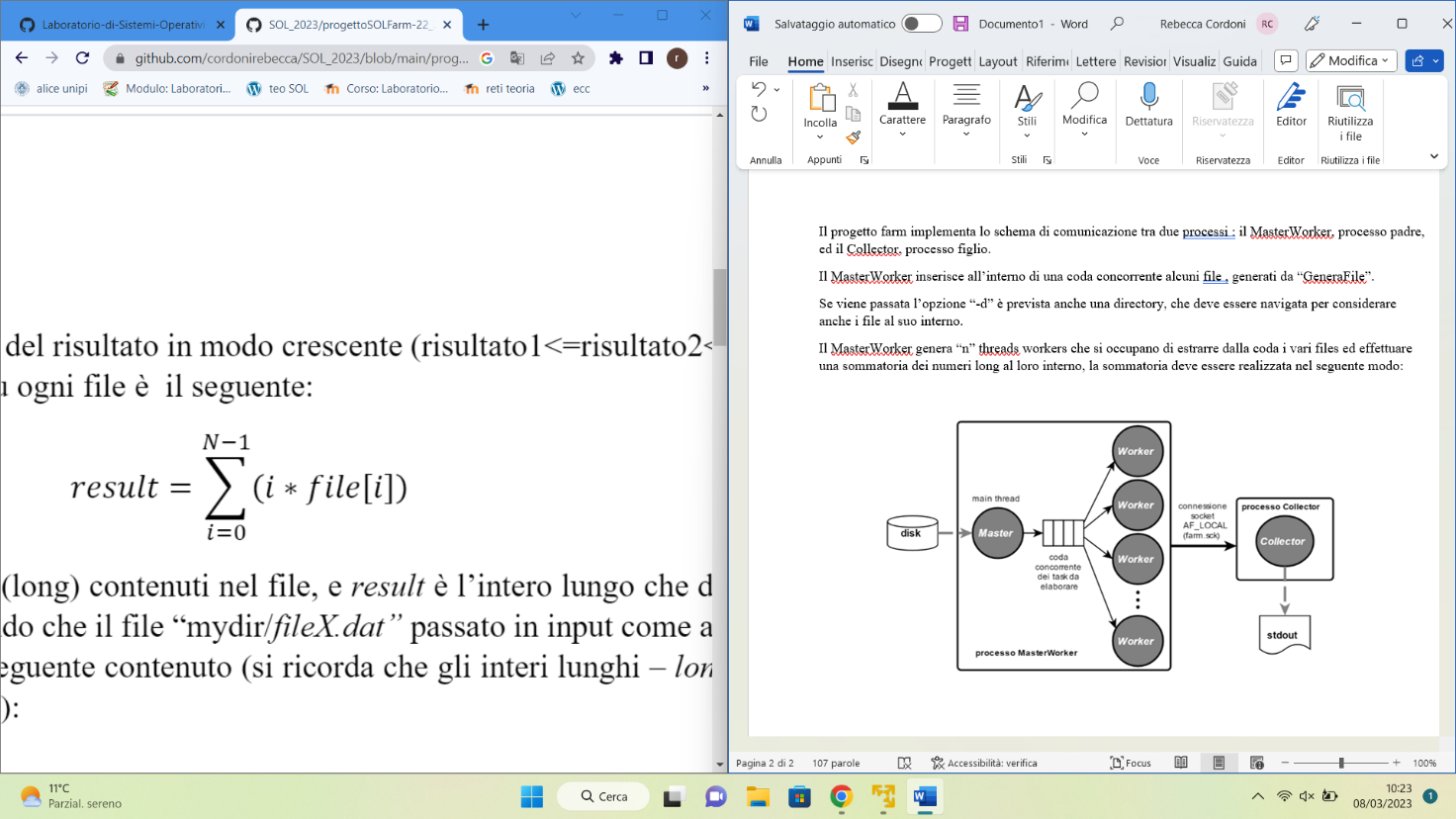
Il progetto farm implementa lo schema di comunicazione (descritto dall’immagine sottostante) tra due processi : il MasterWorker, processo padre, ed il Collector, processo figlio.



I due processi comunicano tramite una connessione socket AF\_LOCAL (AF\_UNIX).

Il MasterWorker inserisce all’interno di una coda concorrente alcuni file , creati da “GeneraFile”.

Se viene passata l’opzione “-d” è prevista anche una directory, che deve essere navigata per considerare anche i file al suo interno.

Il MasterWorker genera “n” threads workers che si occupano di estrarre dalla coda i vari files, ed effettuare una sommatoria dei numeri long al loro interno, la sommatoria deve essere realizzata nel seguente modo:

dove N è il numero di interi long contenuti nel file, e result è l’intero long che dovrà essere inviato al collector assieme al path che indetifica il file.

Il processo MasterWorker deve gestire anche i segnali SIGHUP, SIGINT,SIGQUIT,SIGTERM e SIGUSR1; mentre il processo Collector deve ignorarli.

**Master Worker**

Il processo Master Worker è chiamato nel progetto come “farm”.

È composto dalle seguenti funzioni:

1. **Static void\* sigHandler\_func (void\*arg)**: è la funzione che si occupa della gestione dei segnali, infatti è composta da uno switch che identifica tutti i segnali che devono essere catturati dal Master Worker.

I segnali SIGHUP, SIGINT,SIGTERM,SIGQUIT pongono la variabile *termina* = 1, e in tal modo viene segnalato alla funzione Producer di interrompere l’inserimento di file all’interno della coda concorrente.

1. **Void\* Producer** **(void\* arg)** : è la funzione che si occupa di inserire i vari file nella coda concorrente “q”, con la funzione “push”.

I file che vengono inseriti, sono passati al Producer tramite una lista “l”, dalla quale viene estratto un elemento alla volta, e una volta inserito in “q” viene eliminato dalla funzione “canc”.

1. **Void\* Consumer (void\* arg)** : è la funzione che si occupa di togliere i file dalla coda “q”.

La funzione rappresenta i workers che una volta estratto il file, effettuano una sommatoria dei long all’interno del file, con la funzione “sommatoria” e inseriscono il risultato, ovvero *sommatoria\_risultato* unito al nome del file, con eventuale percorso file nel caso in cui il file sia all’iterno di una directory, in una varibile “path\_socket” .

A questo punto ogni worker accetta una connessione di tipo AF\_UNIX col collector e invia con la funzione “write” il risultato che il Collector dovrà stampare.

Prima di accetttare la connessione c’è una variabile “ termina\_prima” che viene settata a 1 nel caso in cui sia arrivato il segnale SIGUSR1 che determina la chiusura istantanea della comunicazione ed il collector stamperà solo i risultati ottenuti fino a quel momento.

1. **Void\* parser(int argc, char\*argv[], llist\*\* list\_to\_insert)** : è la funzione che si occupa della gestione degli argomenti che vengono passati opzionalmente da riga di comando.

Tra gli argomenti che vengono passati alla funzione c’è List\_to\_insert, che inserisce i file passati da riga di comando all’interno della lista, affinchè vengano poi uniti a quelli dentro le cartelle.

Gli argomenti sono i seguenti:

1. *-n <nthread>* specifica il numero di thread Worker del processo Master Worker (valore di default 4).
2. *-q <qlen>* specifica la lunghezza della coda concorrente tra il thread Master ed i thread Worker ( valore di default 8).
3. *-d <directory-name>* specifica una directory in cui sono contenuti i file binari ed eventualmente altre directory.
4. *-t <delay>* specifica il tempo in millisecondi che intercorre tra L’invio di due richieste successive ai thread Worker da parte del master Worker ( valore di default 0).
5. **Main** : all’interno del main abbiamo il cuore del programma, all’inizio viene chiamata la funzione *parser*, che come abbiamo detto precedentemente si occupa di leggere i comandi dalla linea di comando, presenti nel test.sh.

La funzione *parser* inserisce nella lista List\_to\_insert i file che sono fuori dalle cartelle.

Successivamente viene inizializzata la maschera dei segnali e il segnale SIGPIPE viene opportunamente ignorato dal processo MasterWorker.

Adesso con la funzione “listdir” viene aperta la cartella “testdir” e tutte le cartelle al suo interno, aggiungendo alla lista precedente, List\_to\_insert, i file che erano all’interno di queste cartelle.

A seguire viene effettuata la fork che genera il processo figlio *Collector*.

Il Collector ignora tutti i segnali che erano invece gestiti dal Master Worker e crea un thread che si occuperà della comunicazione socket con i vari workers.

Terminato il ramo della fork dedicato al Collector, si accede al ramo che identifica invece il *Master Worker*, all’interno del quale si effettua la creazione del thread Producer e dei vari Workers ( consumers) .

**Collector**

Il collector è composto da tre funzioni:

1. **Void cleanup()** : è una funzione che si occupa di pulire il socket file nel caso esista precedentemente.
2. **Void\* socket\_collector (void\* arg)** : è la funzione che si occupa di aprire una comunicazione socket con ogni worker, ciò è realizzato dalle funzioni “socket”, “bind”, “listen”, “accept” . I risultati inviati dai workers vengono inseriti in una lista : *List\_to\_stamp* e successivamente questa viene ordinata con il meccanismo del *Bubble Sort*, ovvero ogni coppia di elementi adiacenti viene comparata ed i i due elementi vengono invertiti di posizione se sono nell’ordine sbagliato. Al termine, quando tutti i file vengono inviati, viene stampata la lista con tutti i valori numerici affiancati dal nome del proprio file.
3. **Void bubbleSort(file\_structure \*start)**: è un meccanismo di ordinamento dei numeri in modo crescente.

**Workers**

I workers, come già detto in precedenza si occupano di prendere i file ed effettuare una sommatoria dei numeri long al loro interno.

La funzione utilizzata dai workers per effettuare il calcolo si chiama “sommatoria”.

All’interno di questa si apre il file con la funzione “fopen”, si legge il file aperto e si inseriscono gli interi long all’interno di una lista: List\_file, che ha la seguente struct:

*typedef struct list\_integer{*

*long info;*

*struct list\_integer \*next;*

*}list\_integer;*

si tratta di una semplice lista di interi di tipo long .

scorro poi tutta la lista ed eseguo la sommatoria nel modo descritto dalla traccia del progetto, ed infine chiudo il file.

**List**

List contiene tutte le funzioni utilizzate per gestire le liste:

1. **Queue\_t \*initQueue ()** : è utilizzata per inizializzare la coda q. la coda q ha la seguente struct:

*typedef struct Queue {*

*llist \*head;*

*llist \*tail;*

*unsigned long qlen;*

*pthread\_mutex\_t qlock;*

*pthread\_cond\_t qcond;*

*} Queue\_t;*

1. **deletQueue(Queue\_t \*q)** : è utilizzata per liberare la memoria occupata dalla coda.
2. **Void\*dequeue(Queue\_t\* q)**: è utilizzata per ritornare il primo elemento della coda concorrente e viene anche cancellato.
3. **Int push(queue\_t\* q, void\* data)**: è utilizzata per inserire in modo concorrente tutti i file, di fatti abbiamo la lock prima dell’inserzione dell’elemento e poi una unlock e l’invio del segnale che avvisa che è stato inserito un elemento.
4. **Void Delete\_head\_lista\_piena (struct llist\*\* head, char\* data)**: viene utilizzata nel producer per eliminare il primo elemento della lista se uguale al parametro che gli passo “data”.
5. **Void listidir(const char \*name, struct llist\*l)**: viene utilizzata per aprire tutte le cartelle data una directory e inserire i file trovati nella lista “l”, questa lista verrà poi utilizzata per inserire i vari file nella coda concorrente “q” dal producer;
6. **Void look\_for\_file(char\* filename, char\* directorydipartenza, struct llist\*l)**: questa funzione dato il nome del file e la directory di partenza riporta tutto il percorso file necessario a raggiungere un determinato elemento.
7. **Void insert\_list(struct llist\*\* head, char\*opzione)** : è la funzione utilizzata per inserire i char all’interno delle varie liste create.

La struct di llist è la seguente:

*typedef struct llist{*

*char\* opzione;*

*struct llist \*next;*

*} llist;*

1. **Void canc(llist\*\* head)**: viene utilizzata per cancellare dalla lista tutti gli elementi ecceto l’ultimo.
2. **Void add\_list\_flag(struct llist\*\* headm char\* opzione, char\*var)**: viene utilizzata per inserire un flag alla fine dei file che non sono all’interno della riga di comando ma che magari si trovano in una cartella. In tal modo si possono distinguere da quelli che vengono inseriti da linea di comando e quindi che necessitano di un percorso file per essere aperti.
3. **Int listLenght(llist \*item)** : funzione che calcola la lunghezza di una lista.
4. **Void insert\_integer(list\_ineteger\*\*l, long el)** : è utilizzata per inserire in una lista un insieme di interi long.
5. **File\_structure \*split\_file(llist\* l, file\_structure\* head)**: è una funzione utilizzata per separare la parte numerica da quella letterale in modo che con la funzione bubble sort possa ordinare in modo crescente i valori.
6. **Void insert\_file(file\_structure\*\* head, char\*opzione, long num)**: è utilizzata per salvare il risultato che deve essere stampato dal collentor nella sua parte numerica e quella letterale.

La struct del file è al seguente:

*typedef struct file\_structure{*

*long value;*

*char\* info;*

*struct file\_structure\* next;*

*}file\_structure*;

1. **Void removeChar(char\* str, char chatToRemmove)**: è una funzione utilizzata per eliminare il flag che era stato inserito dal add\_list\_flag.