Relazione sul progetto del modulo di laboratorio SOL 2014/15

Contents:

1 Introduzione

Nella presente relazione vengono discusse le varie scelte progettuali e la loro implementazione (molto a grandi linee…). Ho provato a fare uno schema per illustrare il funzionamento della mutua esclusione, tuttavia non sono convinto della sua chiarezza.

2 Strutture dati principali

Nel progetto ho utilizzato sono una struttura ausiliaria chiamata “workingset” contiene i lavori dei thread worker ed una struttura “motion” utilizzata dalle rules per stabilire lo spostamento del pesce/squalo.

workingset ha due campi:

int first -> prima riga su cui un worker deve lavorare
int last -> ultima riga su cui un worker deve lavorare
possibili valori: [from 0 to nrow]

motion ha tre campi:

char pos -> informazione sulla direzione dello spostamento
possibili valori:
L -> se mi sposto a sinistra
U -> se mi sposto in alto
R -> se mi sposto a destra
D -> se mi sposto in basso

char val -> informazione sul contenuto nella cella in cui mi sto spostando
possibili valori:
F -> se c'è un pesce
S -> se c'è uno squalo
W -> se è presente dell’acqua

int point -> valore della coordinata

3 Strutturazione del codice

In generale preferisco sempre utilizzare molti file con poche righe di codice all’interno (nei limiti del ragionevole), ritengo molto efficiente questo modo di lavorare poiché dando nomi significativi ai file creo una sorta di indice che rende più facile trovare le varie porzioni di codice. Seguendo questo approccio il progetto é suddiviso nei seguenti file:

Libreria wator:

- **wator.h:** header “traccia”, fornito dai docenti
- **rules.h:** header contenente le funzioni e strutture dati di supporto per l’aggiornamento del pianeta.
- **wator.c:** Implementazione delle varie funzioni del wator.h .
- **rules.c:** Implementazione delle regole per l’aggiornamento del planet.
- **rules_function.c:** Implementazioni delle funzioni ausiliarie alle regole di aggiornamento.

Processi wator e visualizer:

- **declaration.h:** tutte le dichiarazioni di strutture dati, funzioni e variabili condivise utilizzate nella scrittura del secondo frammento.
- **extern_variables.c:** inizializzazione delle variabili condivise dichiarate nell'header.
- **aux.c**: funzioni ausiliarie, di secondaria importanza, utilizzate solo per raggruppare il codice e renderlo più ordinato.
- **handler.c:** codice del thread handler, il quale gestisce tutti i segnali.
- **dispatcher.c:** codice del thread dispatcher.
- **worker.c**: codice dei thread worker.
- **collector.c**: codice del thread collector.
- **main.c:** main del processo wator.
- **visualizer.c:** main del processo visualizer.

Con i file del primo frammento seguendo le regole del makefile viene creata una libreria, con gli altri file no.

4 Scelte progettuali

Il progetto è stato realizzato secondo specifica, a parte qualche eccezione concordata con il docente. La gestione degli errori a volte è stata fatta su gruppi di funzioni, poiché in alcuni casi non è importante sapere quale system call ha fallito ma bisogna semplicemente terminare.

Libreria wator

La libreria è stata implementata seguendo le specifiche dell'header. La parte un po' più difficile da realizzare è stata il movimento randomico il quale è stato implementato mediante l'ausilio di un array di strutture motion, con una funzione che lo inizializza ed un altra che effettua l'accesso.
Per evitare che un elemento venga aggiornato più di una volta ho implementato una matrice contenente i vari flag di aggiornamento, che vengono testati per controllare se l'elemento è stato già aggiornato.
Ho modificando la struct planet per aggiungere la matrice dei flag di aggiornamento (upadated), ed ho cambiato il tipo delle matrici btime e dtime le quali prima erano int, ma io ho pensato che il tipo int di default è lungo 4 byte (poi varia da architettura ad architettura ma in media è così) ed ho ritenuto improbabile utilizzare quei valori settati a 2Gigachronon oppure ad un valore negativo, quindi ho utilizzato un tipo più piccolo ovvero l'unsigned short int, che è sempre positivo ed occupa la metà dello spazio degli int. Per la matrice di aggiornamento ho utilizzato il tipo unsigned char (che è un byte) invece del usuale tipo int sempre per gli stessi motivi. Ho valutato di utilizzare un bit direttamente nel tipo cell_t invece di utilizzare una matrice di flag (gestendolo con le operazioni bit a bit) ma ho notato che in spazio risparmiavo circa il 15% dell'occupazione attuale, ma il tempo impiegato dai worker per completare un aggiornamento aumentava del 30% dovendo fare vari shift ed and ed or bit a bit, quindi ho pensato che non valesse la pena utilizzare questo sistema. (maggiori info nei dettagli implementativi).

Processo wator

Questo processo è stato implementato seguendo le specifiche, con due differenze sostanziali, l'aggiornamento della matrice viene effettuato per righe, poiché l'aggiornamento per rettangoli è meno efficiente. Il bordo delle sottomatrici viene considerato sezione condivisa, perché un pesce/squalo, può tranquillamente passare da una sottomatrice all'altra. Secondo questa logica, gli angoli dei rettangoli sarebbero una sezione condivisa tra 3 thread di conseguenza può capitare che ci sia un worker che aggiorna l'angolo e due thread in attesa. Mentre aggiornando per fasce la

sezione condivisa è sempre al massimo tra due thread. Il main ha il compito di settare la maschera dei segnali, allocare le strutture dati, creare il processo visualizer, i thread handler, collector, dispatcher ed i vari workers. La gestione dei segnali viene effettuata tramite la sigwait ed un thread che si sospende su di essa, ho deciso di utilizzare questa funzione perché non è un handler, in questo modo non ho sezione critica e vincoli sulle funzioni utilizzabili. Oltretutto all'arrivo un segnale se mascherato non interrompe le system call e non devo preoccuparmi di farle ripartire se non avevano terminato l'esecuzione. La comunicazione con il visualizer avviene tramite le socket. La matrice al fine di minimizzare i byte trasmessi viene compressa in una bitmask ed inviata. Poiché non è necessario ricalcolare i lavori ad ogni chronon, il dispatcher genera i lavori li mette in una coda FIFO (implementata tramite un array di workingset) e termina. Il lavoro acquisito dai worker dipende dall'ordine con cui vengono schedulati e non è deterministico. Il collector crea un ulteriore thread che si occupa di inviare la matrice al visualizer, così mentre un thread comunica con il visualizer, il collector si occupa di far ripartire i worker a mano a mano che terminano.

Processo visualizer

Questo processo interagisce con il processo wator, tramite le socket. Il visualizer è il lato server della socket, single-thread che accetta una sola connessione (sperabilmente da parte del wator) riceve dati e stampa. Non è prevista la ricezione di più connessioni.

5 Struttura del programma

Libreria wator

Le uniche parti dell'implementazione più complesse ed interessanti da discutere sono l'implementazione delle regole di aggiornamento degli elementi della matrice.
L'implementazione è simile per tutte le regole, l'idea di fondo comune a tutte è questa:

1. Scansione delle possibili posizioni in cui l'elemento si può spostare (o riprodurre) con salvataggio delle informazioni nella struttura motion;
2. Scansione della struttura motion e restituzione del numero dei possibili spostamenti;
3. Scelta a caso di uno spostamento (se possibile);
4. Spostamento dell'elemento

Processo wator

Questo è il processo principale di tutto il progetto. Il funzionamento di questo processo è abbastanza semplice e lineare.

1. Appena avviato imposta la maschera dei segnali.
2. Crea il thread gestore dei segnali.
3. Alloca le strutture dati in funzione dei dati in ingresso.
4. Crea gli altri thread ed il visualizer.
5. Aspetta la terminazione dei vari thread.
6. Libera lo spazio di lavoro e termina.

Thread handler

Questo thread resta sempre sospeso, si sveglia solo all'arrivo di un segnale, lo gestisce, e si sospende oppure se necessario termina.

Thread dispatcher

Questo thread analizza il pianeta in ingresso ed il numero di thread worker, suddivide la matrice nelle varie sottomatrici e inserisce tutto nella coda FIFO. Poi termina.

Thread worker

Questo thread viene attivato dal collector e:
1. Accede alla coda FIFO in mutua esclusione e controlla se ci sono lavori disponibili, se non sono presenti si sospende.
2. Nella coda FIFO trova una struttura contenente due indici, li analizza, se sta aggiornando la sezione condivisa con altri worker tenta di acquisire la mutua escluzione, aggiorna la riga, ed esegue il controllo, fino a terminare l'aggiornamento.
3. Notifica al collector di aver terminato l'aggiornamento e si sospende.

Thread collector

Questo thread appena passa in esecuzione:
1. Stabilisce la connessione con il visualizer.
2. Comunica il numero di righe e colonne al visualizer
3. Crea il thread writer.
4. Controlla se i worker hanno terminato l'aggiornamento, se necessario si sospende
5. Controlla se deve comunicare con il visualizer, altrimenti riattiva i worker e riprende dal punto 4.
6. Bufferizza e codifica la matrice.
7. Riattiva i workers.
8. Riattiva il writer.
9. Si sospende fino a quando i workers non terminano.

Thread writer

Questo thread viene creato dal collector.
1. Controlla se deve comunicare con il visualizer, altrimenti si sospende.
2. Invia la matrice al visualizer.
3. Notifica il completamento al visualizer, e si sospende.

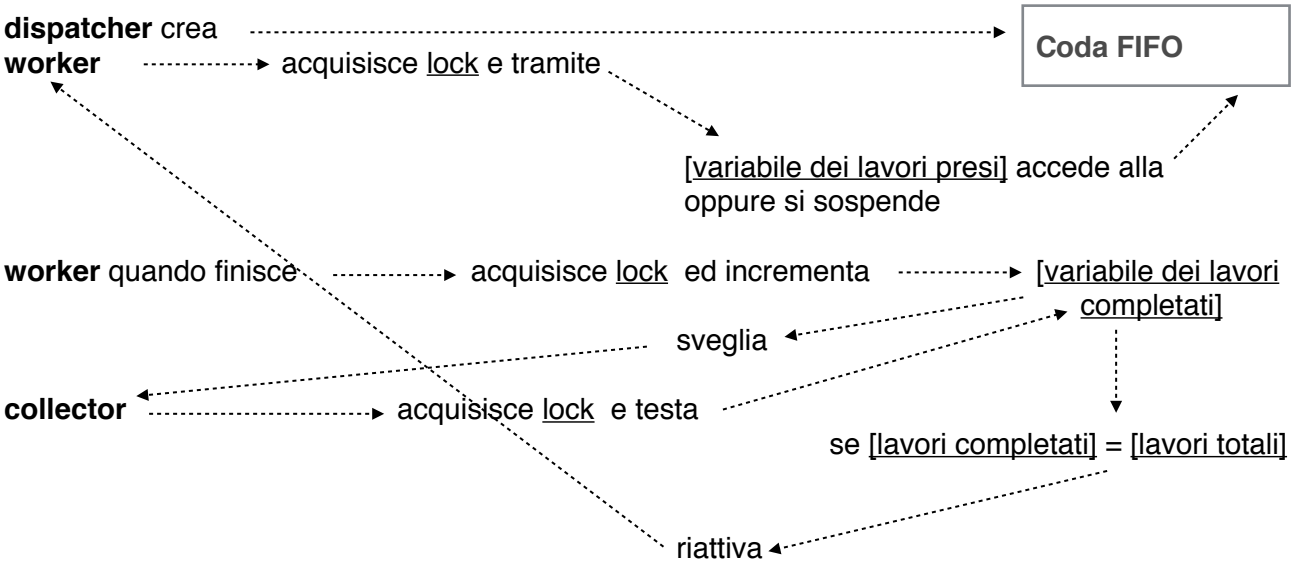
Processo visualizer

Questo processo viene creato dal wator e:
1. Crea la socket.
2. Attende la connessione del wator.
3. Riceve la matrice dal wator.
4. La decodifica e stampa.
5. Torna al punto 2 oppure termina.

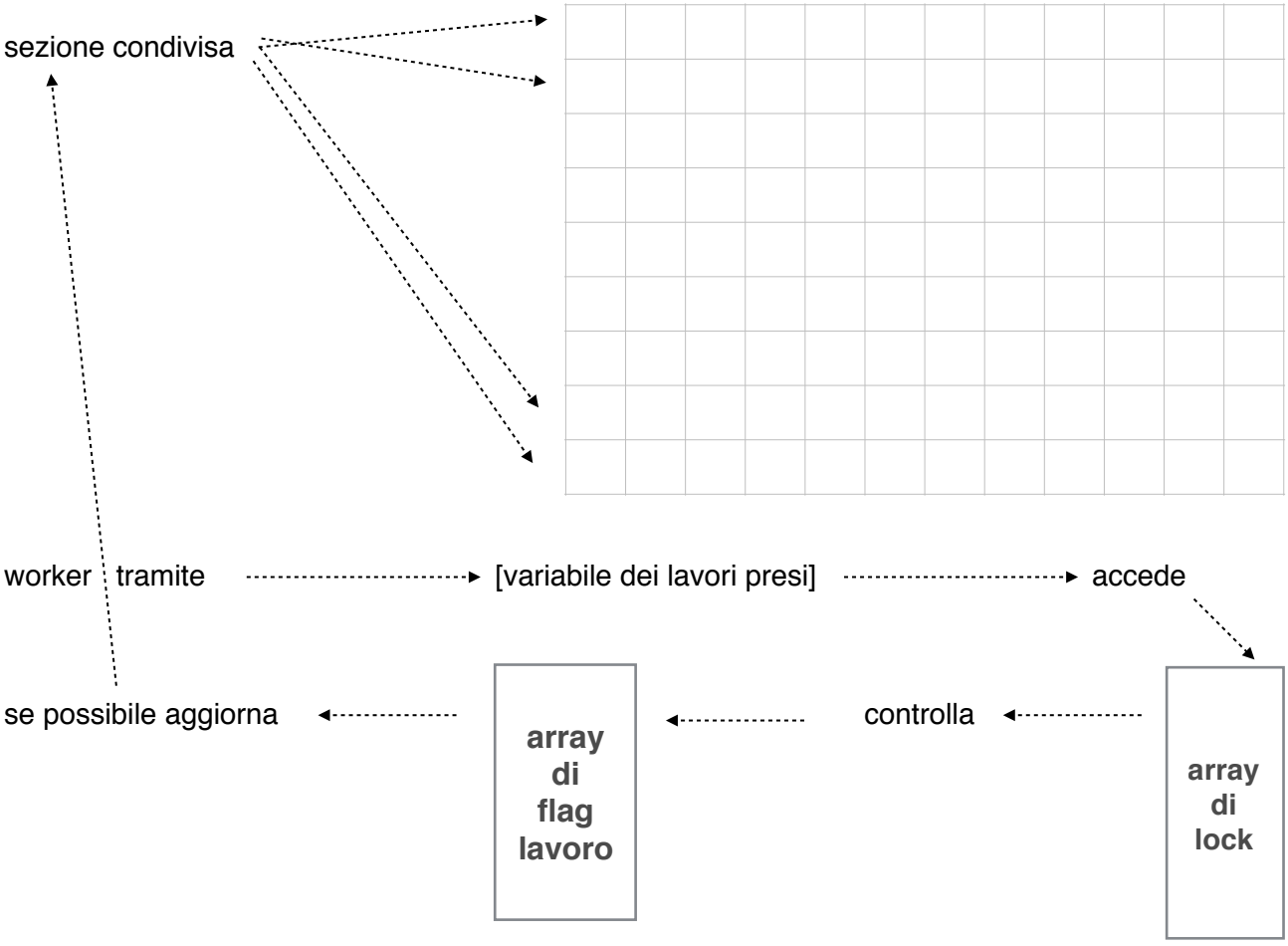
6 Gestione della mutua esclusione

Le strutture dati condivise a cui si deve accedere in mutua esclusione sono la coda FIFO ed alcune sezioni del pianeta.
Ho utilizzato una lock per accedere alla coda FIFO ed una variabile di condizione che causa la sospensione dei worker se tutti i lavori sono presi. Per accedere alle sezioni della matrice condivise (il bordo però si intende di due righe per lato non una), si utilizzano più lock, ogni sezione

condivisa ha la sua lock e la sua variabile di condizione. Il worker che vuole accedere ad una sezione condivisa deve acquisire la lock corrispondente a quel bordo, controllare che l'altro worker non stia aggiornando quella sezione, aggiornare la sezione e rilasciare la lock. Quando un worker termina l'aggiornamento accede in mutua esclusione ad una variabile che tiene il conto dei lavori ultimati, la incrementa, e sveglia il collector. Il collector per risvegliare i worker accede in mutua esclusione alla variabile dei job presi, e la resetta, poi si sospende.



Sezioni condivise nell'aggiornamento per riga della sottomatrice:



Utilizzo un array di lock per ogni sezione condivisa così non si sospendono tutti i thread ma solo i thread che condividono la sezione. In caso di aggiornamento di una sezione non condivisa, il worker salta tutti i controlli non acquisisce alcuna lock ed aggiorna direttamente

7 Protocollo di comunicazione

Il protocollo di comunicazione è basato su socket AF-UNIX. La comunicazione è monodirezionale, wator -> visualizer e non viceversa. Il wator per minimizzare il numero di byte spediti comprime la matrice, essendo il pianeta formato solamente da tre elementi, sono necessari due bit per indicare un elemento. Il wator quindi sostituisce il tipo cell_t con una maschera di bit, risparmiando così sei bit per ogni elemento spedito. Una volta codificata la matrice viene effettuata un'unica write verso il visualizer (per minimizzare l'overhead della write). Il visualizer decodifica e stampa. La terminazione del visualizer è gestita tramite una variabile spedita dal wator (end).

8 Watorscript

Un semplice script realizzato in bash per il checking dei file planet.
La sua realizzazione è stata abbastanza semplice, viene aperto il file, vengono letti gli interi nrow ed ncol (viene controllato che siano interi) poi utilizzando questi interi si entra in un ciclo in cui si chiama la read (viene impostata la variabile IFS a vuoto, in modo da non skippare alcun carattere) nrow*ncol/2 volte e si testa se la stringa letta è valida.