RELAZIONE FINALE PER IL PROGETTO WATOR

Laboratorio di sistemi operativi

Università di Pisa – a.a. 2014/2015

A cura di Simone Schirinzi

L'intero progetto si compone di 7 file.

- wator.c e wator.h implementano la parte relativa alla gestione del wator.
- auxfun.c e auxfun.h contengono funzioni ausiliari per la gestione della coda e per la comunicazione tra i processi.
- watorprocess.c implementa il processo principale.
- visualizer.c implementa la resa a video del wator.
- watorscript consente il controllo di un file pianeta.

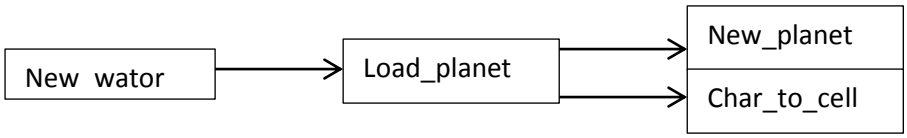
LA LIBRERIA WATOR

Un wator è una struttura composta da un pianeta, di dimensione nrow*ncol, e da alcuni valori rappresentanti l'aspettativa di vita e l'età fertile per squali e pesci.

Per inizializzare un wator bisogna invocare la funzione new_wator, dando in input una configurazione iniziale, e un file contente i valori di nascite e morti denominato wator.conf .

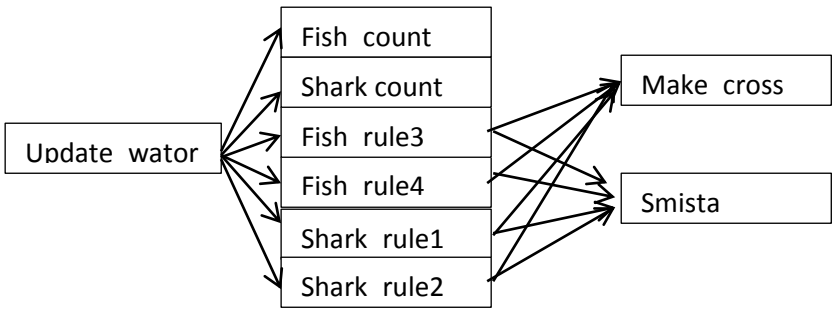
La funzione, dopo aver controllato l'effettiva esistenza e consistenza dei file sopra descritti, crea un nuovo ecosistema, all'interno del quale viene creato un nuovo pianeta, inizialmente solo con acqua.

Il file planet viene letto carattere per carattere. In presenza di una S (squalo) o di una F (pesce), si inserisce SHARK o FISH nella posizione corrispondente della matrice. Ovvero si effettua una conversione da char in cell_t.



Wator vive con update_wator. La funzione, per ogni chronon, si crea due vettori rappresentanti gli animali da muovere (per evitare di muovere due volte), e conta quanti squali e quanti pesci ci sono.

Update_wator applica le 4 regole presenti nella simulazione, facendo prima muovere i pesci e poi gli squali.



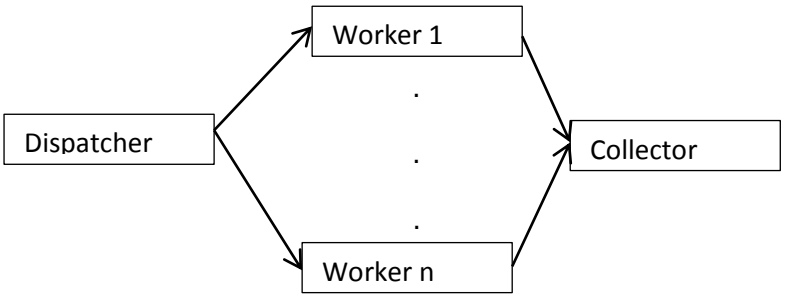
Il valore di chronon è incrementato solo alla fine dell'applicazione di ogni regola a ogni animale (eventualmente sopravvissuto).

Ho scelto per questo diverso ordine, rispetto a quello proposto, perchè evito che vengano mangiati pesci che potrebbero essere pronti per riprodursi.

IL PROCESSO WATOR

Il processo ha il compito di gestire un wator in un ambiente multithread.

Il wator viene caricato normalmente, dopodiche una struttura farm gestisce l'aggiornamento.



Il wator viene aggiornato in modo simile a quanto avveniva con la funzione update_wator, tuttavia l'aggiornamente viene eseguito da n worker, che elaborano ognuno una sottomatrice del pianeta.

Quando ogni worker ha terminato, viene incrementato il valore di chronon.

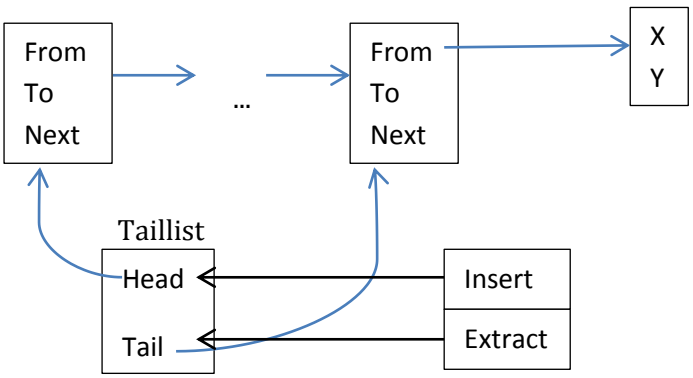
Il thread Dispatcher ha il compito di inserire le sottomatrici da elaborare in una coda condivisa.

Il dispatcher inserisce nella coda preferibilmente sottomatrici di dimensione 5 x ncol, tenendo però in considerazione sia se ci sono abbastanza worker, sia se nrow non è multiplo di 5.

La coda condivisa "taillist" è una lista di triple <From,To,Next>, dove From e To sono dei punti (delle coppie di valori <x,y>) e Next è il puntatore all'elemento successivo della lista. Per accedervi vi sono 2 puntatori , rispettivamente al primo e all'ultimo elemento della lista, <Head,Tail>, sui quali operano le funzioni Insert e Extract, per l'inserimento in coda, e l'estrazione in testa. Quest'ultime sono implementate in auxfun.c .

L'accesso alla coda avviene in mutua esclusione, tramite un variabile pthread_mutex_t .

Le sottomatrici sono rappresentate dalle coordinate del punto in alto a sinistra e da quello in basso a destra



Ad ogni inserimento, viene inviato ad ogni worker un segnale per iniziare l'elaborazione della tile.

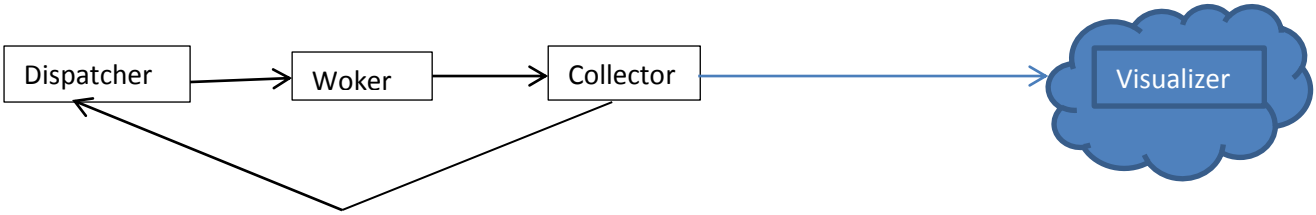
Il worker estrae un elemento dalla coda, lo elabora, e invia un segnale al Collector.

Come da specifica ogni worker crea, appena avviato, un file vuoto wator_worker_x, dove x viene sostituito dal proprio numero identificativo, ovvero un valore compreso tra 0 e il numero di worker.

Ogni worker elabora la sottomatrice in modo concorrente agli altri worker, interagendovi. Al suo interno conserva due vettori contenenti le posizioni degli animali da muovere. Essi vengono svuotati alla fine di ogni elaborazion, e presi dal wator all'inizio di quella successiva. Una volta per chronon. Per evitare di prendere animali già mossi da altri worker, quest'ultimi si sincronizzano tra di loro prima di iniziare l'elaborazione, tramite una variabile pthread_cond_t.

Quindi ogni worker applica in sequenza, al proprio insieme , e in mutua esclusione, le quattro regole del wator.

Il collector si sveglia non appena tutti i worker che avevano estratto un elemento dalla coda, hanno terminato la loro elaborazione. Invia al visualizer un messaggio contenente il pianeta da stampare e segnala al dispatcher di iniziare una nuova elaborazione



Il collector ha anche il compito di inizializzare la socket, e di connettersi a quella creata dal visualizer.

IL PROCESSO VISUALIZER

Visualizer si comporta come server nel confronti di wator.

Crea una socket AF_UNIX visual.sck, e si mette in ascolto di comunicazioni da wator.

Ricevuto il pianeta, esso provvede a stamparlo a video o su file, a seconda delle indicazioni.

COMUNICAZIONI TRA WATOR E VISUALIZER

Le comunicazioni hanno lunghezza fissa di 1024 byte, per motivi di efficienza.

La comunicazione del pianeta avviene tramite un unica stringa (eventualmente inviata a segmenti di 1024 caratteri). Le funzioni per la conversione sono implementate nella libreria auxfun.

Esse visitano le matrice cella per cella, convertendo tutto in caratteri con cell_to_char. Il carattere "." segnala, nella stringa, la fine di una riga.

Appena viene effettuata la comunicazione, wator invia in sequenza 5 parametri, per consentire al visualizer di stampare correttamente il pianeta. Ovvero:

- Dimensione della stringa-pianeta
- Posizione del file su cui stampare il pianeta
- Dimensione verticale del pianeta
- Dimensione orizzontale del pianeta
- Valore opzionale

Infine , la stringa-pianeta, ricevuta e stampata infinite volte, o fino alla ricezione di SIGINT o SIGTERM.

GESTIONE DEI SEGNALI

In entrambi i processi sono gestiti in modo non automatico i segnali di SIGINT, SIGTERM, SIGPIPE. Wator inoltre gestisce SIGUSR1 per il salvataggio del pianeta.

SIGPIPE è ignorato.

SIGTERM e/o SIGINT settano una variabile globale di tipo sig_atomic_t. Stessa cosa SIGUSR1.

Wator usa SIGUSR1 per il salvataggio del pianeta su wator.check.

Entrambi usano la variabile settata da SIGINT e/o SIGTERM per effettuare una terminazione gentile.

Wator controlla, alla fine del loop del collector, il valore della variabile di terminazione. Invia al visualizer il segnale di terminazione e attende che termini. Chiude la socket e setta un altra variabile per la terminazione dei worker e del dispatcher. Attende che terminino anche loro, quindi chiude il processo.

Visualizer controlla, alla fine del main loop, il valore della variabile di terminazione. Quindi chiude la socket e termina il processo

WATORSCRIPT

Watorscritp è un bash script per controllare se un file è un file pianeta, ed eventualmente contarne il numero di squali e/o di pesci al suo interno.

Con l'opzione –help, stampa un messaggio di aiuto.

Scrivendo solo il file da controllare, si ottiene OK o NO come risposta, a seconda se sia ben formattato o meno o si sia verificato un errore. Lo script infatti legge il file riga per riga.

Le prime 2 righe devono contenere due numeri (righe e colonne della matrice).

Le altre righe vengono formattate a sequenze di caratteri.

Quindi si controlla che :

- la stringa abbia la stessa dimensione del numero di collone.
- Vi siano esattamente numero di righe stringhe.
- Ogni stringa contenga esclusivamente i caratteri W, F, o S.

Con l'opzione –s si ottengono il numero di squali, mente con –f quello dei pesci.