Progetto SO 2020/21

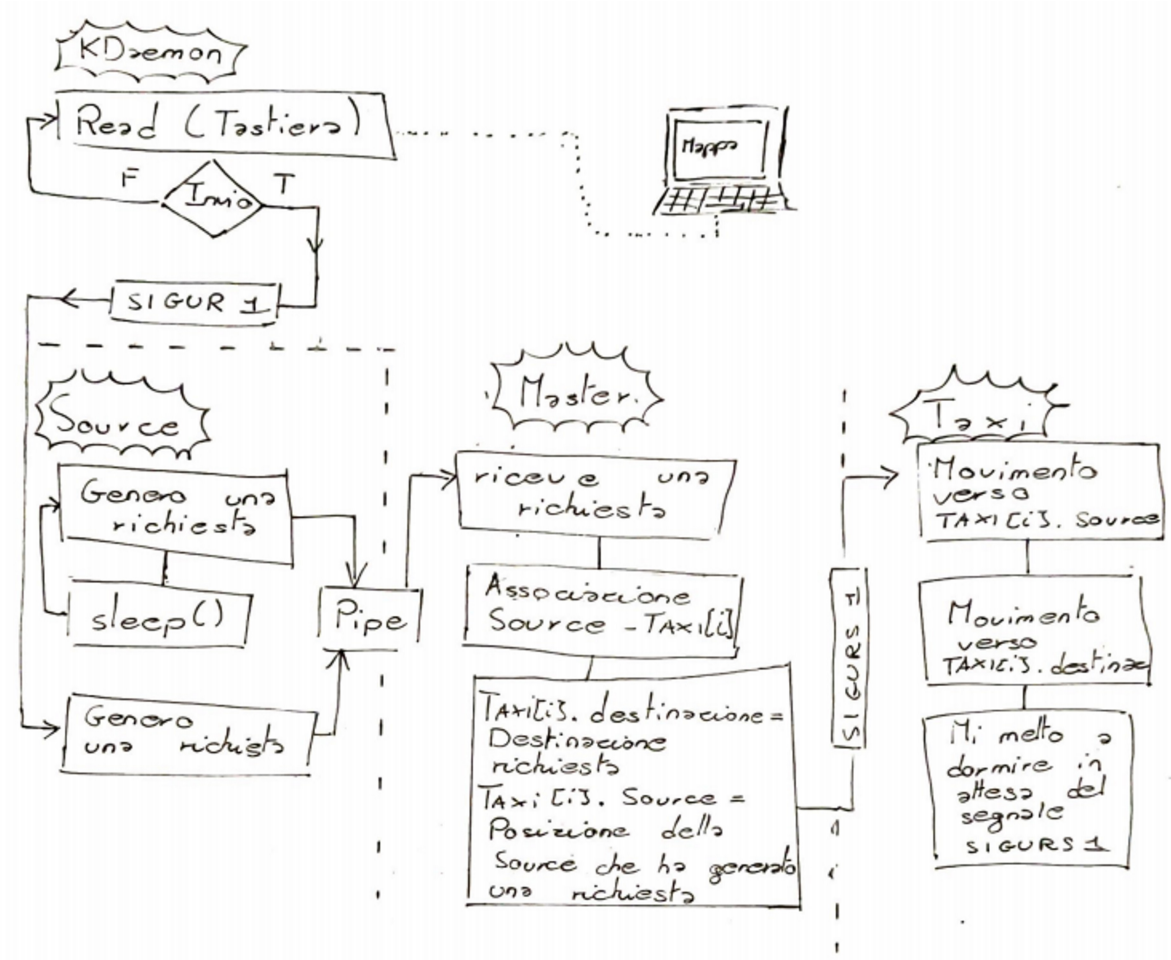
The Taxicab game

Enrico Chiesa, Laurentiu Apostol

12 gennaio 2021

Indice

1. Schema grafico



2. Struttura dati

Memoria condivisa

- ### Mappa
  Ogni cella è rappresentata dalla ***struct*** `cella`.
  Ogni cella contiene dati correlati alla singola casella della mappa come SO_CAP,SO_TIMENSEC.
  In particolare abbiamo scelto di inserire dei campi all'interno della struct che ci dicessero:
  - Se la casella è un HOLE --> occupata==1;
  - Se la casella è una SOURCE --> SOURCE==1;
  - Il numero di taxi attualmente all'interno --> cur_num_taxi;
  - Il numero di taxi che hanno attraversato la casella --> tot_taxi;

  La mappa è memorizzata come matrice quadrata (**) di `celle` in **memoria condivisa**.
- ### PosTaxi
  Contiene informazioni sui taxi come:
  Posizione del taxi, posizione della SOURCE che gli ha fatto una richiesta, destinazione finale richiesta ed alcuni campi per le statistiche.
- ### Array statistiche
  È un array di due elementi per tenere conto delle corse andate a buon fine e quelle abortite.
  Questi due contatori devono essere memorizzati in una memoria condivisa perché ogni taxi deve poterle aggiornare.

Pipe

Una sola pipe aperta in scrittura da tutti i processi SOURCES ed aperta in lettura dal Master.

Semafori

Due array, lunghi quanto il numero di celle della mappa.
Il primo serve principalmente nel movimento dei taxi (più taxi che si vogliono spostare nella stessa cella).
Il secondo viene utilizzato principalmente per modificare i dati contenenti le statistiche.
Alla fine di questi array sono stati aggiunti, per comodità di utilizzo, rispettivamente tre e due semafori aggiuntivi. Lo scopo di questi semafori verrà illustrato in seguito.

3. Posizionamento iniziale

- **HOLES**
  Sono i primi ad essere posizionati perché il posizionamento di SOURCES e TAXI dipende dagl'HOLES
  Ciclo di posizionamento:

4. Gli indici i,j della posizione di un HOLE vengono generati random;
5. Se la casella è libera* viene contrassegnata come HOLE.
   Altrimenti si ritenta dal punto 1 per *n* volte. *n*=`SO_HEIGHT*SO_WIDTH/2;`
6. Cerco una casella libera in modo sequenziale a partendo dall'ultimo indice casuale i,j.

*libera= Non è stata assegnata ad un HOLE e nelle 8 caselle vicine non ci sono HOLES.

- **SOURCES**
  Stesso ragionamento usato per gli HOLES però:
  In questo caso una casella è "libera" se: non è già stata impostata come HOLE o SOURCE.
- **TAXI**
  Stesso ragionamento usato per gli HOLES però:
  In questo caso una casella è "libera" se non è stata impostata come HOLE e
  cur_num_taxi < tot_taxi .

Errori

Nel caso il programma non trovi una posizione libera in cui posizionare un elemento viene segnalato un errore specifico.

4. La stampa

Mappa

Ogni secondo viene stampata la mappa e le posizioni di tutti i taxi.
Per chiamare la funzione di stampa ogni secondo abbiamo utilizzato al System call alarm(1).

Problema

L'operazione di stampa non è atomica quindi c'è il rischio di stampare dati inconsistenti, nel caso lo scheduler decidesse di interrompere la stampa a metà e lasciare che qualche taxi modifichi la propria posizione o le proprie statistiche globali.

Soluzione

La stampa deve avvenire senza che nessun processo taxi possa alterare la propria posizione.
Per fare ciò, vengono usati due semafori, uno dei quali ha lo scopo di permettere ai taxi di compiere operazioni, quando esso si trova a 0, e che non permette ai taxi di compiere operazioni che alterano lo stato della mappa o delle statistiche finali, nel caso esso venga aumentato di 1 dal master; il secondo semaforo, invece, viene aumentato di 1 dai taxi prima di compiere un'operazione (la richiesta sul primo semaforo e l'aumento di 1 del secondo avvengono insieme atomicamente) e viene diminuito da essi sempre di 1 quando sono usciti dalla loro sezione critica, per segnalare la fine di un'operazione. Il master, quindi, prima di eseguire una stampa, aumenta di uno il primo semaforo e aspetta che il secondo arrivi a 0 (i taxi che ancora stanno svolgendo un'operazione) prima di fare la sua stampa, ed infine diminuisce di 1 il primo semaforo, per permettere di nuovo ai taxi di eseguire operazioni.

5. I processi

Creazione

Per ogni famiglia di processi figli prepariamo un array di stringhe `args` di input con tutti i parametri di cui hanno bisogno.
Al momento della creazione dopo la fork(), nella parte di codice che viene eseguita solo da processi figlio usiamo la `execvp(file_path,args).` In questo modo ogni processo ha solo i dati di cui ha bisogno e non ci sono sprechi di memoria.

- ## Taxi
  - ### Interazione master taxi

    Dopo le opportune inizializzazioni, gestione dei segnali, conversioni degli argomenti in interi, accesso alla memoria condivisa e richiesta sul semaforo, inizializzato al numero massimo di taxi permessi nella cella in cui il taxi si trova, il processo imposta la propria destinazione a -1 (il master l'aveva inizializzata a -2), segnalando in tale modo al master che esso è pronto a ricevere richieste. In particolare, il modo in cui il master invia una richiesta ha due passaggi: il primo è di impostare il campo destinazione della struct, in memoria condivisa, di quel particolare taxi, alla destinazione finale, e il campo della source, nella stessa struct, con la posizione della source da cui è partita la richiesta; il secondo è quello di mandare il segnale SIGUSR1 al processo, che da' il via alla corsa. SIGUSR1 viene gestito dal processo taxi nel seguente modo: all'inizio del programma, il processo aggiunge alla sua maschera di default anche il segnale SIGUSR1, in modo che non possa riceverlo quando non è ancora pronto a farlo. In seguito, imposta la funzione che si occuperà di effettuare la corsa come handler di SIGUSR1. Infine, dopo aver impostato la propria destinazione a -1, esso entra nel suo loop infinito principale, nel quale usa la funzione sigsuspend per entrare in stato di waiting, in attesa che un signal lo risvegli. Oltre a ciò, sigsuspend imposta anche la vecchia maschera, in modo da poter ricevere anche il signal SIGUSR1. Uscito dal waiting, dopo essersi accertato che ad averlo svegliato è stato proprio SIGUSR1, il processo reimposta la sua destinazione a -1 e ricomincia il loop.

Movimento taxi

I taxi si muovono solo orizzontalmente oppure verticalmente, prediligendo l'asse orizzontale rispetto a quello verticale, nel caso sia necessario un movimento in entrambe le direzioni per raggiungere la meta. Inoltre, essi implementano una strategia molto semplice di aggiramento delle buche, strategia influenzata dal vincolo che non ci possono essere due buche vicine (ad esempio, se devo andare a sinistra ed in alto, ma a sinistra c'è un buco, allora sono sicuro di poter andare in alto).
L'implementazione del pathfinder è basata su un'automa a stati finiti, e quindi il risultato, nel codice, equivale ad un'insieme di if/else-if annidati, dentro un ciclo la cui condizione di loop è che la posizione del taxi sia diversa dalla meta (caso in cui ho raggiunto o la source o la destinazione). La funzione che si occupa del raggiungimento della source e della destinazione, cioè il handler di SIGUSR1, sceglie come meta la source, se essa non è uguale a -1, oppure la destinazione, in caso contrario. Raggiunta la meta, se la source era diversa da -1, essa viene impostata a -1. In caso contrario, la funzione termina, e lascia alla funzione principale del main il compito di impostare la destinazione a -1, in modo da segnalare al master la prontezza ad una nuova corsa.
Inoltre, per quanto riguarda il movimento, SO_TIMEOUT è stato interpretato come tempo massimo che un taxi può restare fermo ad aspettare di entrare in una cella, prima di doversi riposizionare. Tale scelta interpretativa è stata fatta sia perché, per qualunque altra operazione, che non può andare in deadlock, non avrebbe senso far riposizionare il taxi, soprattutto se il timeout fosse dovuto a qualche scelta sfavorevole dello scheduler, sia perché, appunto, l'intento esplicito di SO_TIMEOUT era proprio quello di evitare deadlock.

Iterazioni tra taxi

Tutte le **interazioni** tra taxi sono **mediate da semafori.**
Ci sono tre tipi di interazioni che possono avvenire tra i taxi:

a. **Il primo è quello di provare a spostarsi in una casella della mappa.** Per fare ciò, il taxi prima fa richiesta sul semaforo della casella sulla quale vuole spostarsi, e, se tale operazione va a buon fine, esso aggiorna le sue statistiche globali (semafori master-taxi) e poi rilascia il semaforo della cella dalla quale si è spostato, in modo che un altro taxi possa accedervi. Essendo che questo pattern è a rischio di deadlock, alarm viene usato per "svegliare" il processo addormentato sulla richiesta alla casella nella quale voleva spostarsi. Se questo accade, il taxi aggiorna le statistiche globali, mettendo la sua posizione a -1 e diminuendo il nr. dei taxi nella struct della cella nella quale si trovava, in modo da non venire più stampato in tale cella, rilascia la risorsa del semaforo corrispondente sempre a quella cella ed inizia il processo di riposizionamento.

b. **Il secondo tipo di interazione tra taxi è** quello di provare ad **aggiornare una delle due caselle di memoria condivisa**, nella quale si trovano le corse completate e le corse fallite, oppure quello di **provare ad aggiornare alcuni campi della struct della cella**, come quello del numero di taxi presenti in una cella, in un determinato momento, o il numero di taxi che hanno attraversato quella cella. Ci sono, quindi, due semafori di tipo mutex per le due caselle di memoria condivisa, ed un'array di semafori, sempre di tipo mutex, tanti quanto le celle della mappa. Per ogni operazione di modifica di questi dati, quindi, avviene prima la coordinazione Master-taxi, poi la richiesta sui mutex della risorsa che si vuole alterare, il rilascio di mutex, ed infine la segnalazione al Master che le operazioni del taxi sono state completate.

c. I**l terzo tipo di interazione è quello dei taxi che**, a causa di un timeout mentre richiedevano di spostarsi in una cella, **hanno iniziato il processo di riposizionamento**. Per non rischiare che ci siano troppi taxi che provano a riposizionarsi allo stesso momento, rallentando di molto la procedura di riposizionamento, è stato creato un semaforo apposito, inizializzato ad un valore che dipende da SO_CAP_MAX. Esso funge da "limitatore" di quanti taxi possono provare a riposizionarsi contemporaneamente. Il valore di inizializzazione di tale semaforo è stato scelto in maniera empirica, vedendo quali parametri sortivano i migliori risultati (rimuovendo del tutto il semaforo, le prestazioni peggioravano in molti casi).

- ## Stampa

  Dato che le operazioni svolte dai taxi non sono visibili dalla visualizzazione della mappa che viene effettuata ogni secondo abbiamo deciso di far scrivere ad ogni taxi i suoi movimenti su file dedicato all'interno della cartella *taxi_logs*.
  In questo modo è possibile controllare il percorso effettuato da un singolo taxi.

- Gli altri processi

- ## Master

  Il master ha un ciclo che infinito in cui legge la pipe.
  Quando legge la pipe il master **associa** alla cella della source che ha fatto la richiesta un TAXI k libero.
  Scelta del taxi: confrontare tutti i taxi per fare una scelta ponderata impiegherebbe troppo tempo per cui abbiamo deciso di confrontare solamente i primi 10 taxi liberi scegliendo il più vicino alla SOURCE.
  Una volta scelto il taxi a cui assegnare la richiesta vengono aggiornati i campi **source** e **destinazione** della struct Pos che si trova nella posizione, nell'array di memoria condivisa, corrispondente al taxi scelto. In questo momento il taxi ha tutte le informazioni necessarie e viene "svegliato" inviandogli il segnale `SIGUSR1` ricevuto il quale può iniziare la sua corsa.

- ## SOURCE

  Quando una source genera una richiesta scrive sulla pipe due cose:
  1)La destinazione della richiesta.
  2)Il numero della source (il suo indice).

  Le SOURCE generano una richiesta:
  Ogni *n* millisecondi. *n = valore random compresa tra 20 e 120.*
  Oppure quando ricevano il segnale SIGUSR1.
  Dopo aver scritto sulla pipe il processo di "addormenta" utilizzando la `nanosleep()`.

- ## KDaemon

  Quando legge da tastiera il carattere (Invio/Enter) invia il segnale SIGUSR1 a tutte le source.