Relazione del progetto di laboratorio di Sistemi Operativi: The Taxicab Game

Studente: Daniel Haures
Matricola: 919707
Email: daniel.haures@edu.unito.it

Indice

Struttura del codice

Le variabili a tempo di esecuzione sono state implementate tramite variabili d'ambiente caricabili con l'esecuzione di: **source ./ "configurazione" .sh.**
Ci sono tre file di tipo sh, che codificano la configurazione large, la configurazione dense e una configurazione modificabile a piacimento.
Il processo master si occupa della gestione dei processi figli taxi e source, definiti rispettivamente in taxi.c e source.c.
I processi source generano delle richieste di servizio verso i taxi.
Un processo taxi raccoglie e elabora le richieste di servizio ricevute da un processo source.
Tramite l'utilizzo del header common.h ogni processo condivide le stesse macro e ha accesso a un set di funzioni comuni definite nel file support.c
La compilazione e esecuzione del programma è facilitata tramite **makefile** che permette:

- Di eseguire una pulizia dei vecchi file tramite **make clean**.
- Compilare il programma con le macro SO_WIDTH e SO_HEIGHT di configurazione dense tramite **make alld**.
- Compilare il programma con le macro SO_WIDTH e SO_HEIGHT di configurazione large tramite **make all**.
- Eseguire il programma tramite **make run**.

Il progetto è stato sviluppato sulle distribuzioni linux UBUNTU e FEDORA ed è stato provato su più macchine.

Istruzioni di esecuzione

1.Eseguire un eventuale **make clean** per pulire vecchi file
2. Uno dei seguenti comandi per compilare le variabili d'ambiente:

- "source ./largeexport.sh" per configurazione large.
- "source ./denseexport.sh" per configurazione dense.
- "source ./custoumexport.sh" per configurazione personalizzata.

3.Eseguire:

- **make all** per una mappa di grandezza large.
- **make alld** per una mappa di grandezza dense.

4.Utilizzare **make run** per eseguire il programma

L'istruzione 2 può essere effettuata anche dopo la compilazione

Esempio con configurazione dense:

```
make clean
source ./denseexport.sh
make alld
make run
```

Realizzazione della mappa

La mappa è memorizza in una memoria condivisa e contiene all'interno una struct principale contenente a sua volta una matrice di dimensione fissa di struct di tipo cella.
La struttura di tipo cella contiene una variabile info che definisce se la cella è vuota, contiene dei taxi oppure se è una delle celle di blocco.
Una variabile issource che riferisce se la cella in questione è occupata da un processo sorgente.
Invece, le variabili atttime,curtaxi,totatt memorizzano rispettivamente, il tempo di attraversamento della cella, i taxi correntemente presenti sulla cella e il numero totale di attraversamenti sulla cella.
A ogni cella è correlato un semaforo inizializzato con un valore che definisce la capienza di attraversamento della cella.
L'occupazione di una cella da parte di un processo corrisponde al decremento del semaforo della rispettiva cella.
Per la modifica di una singola cella da parte di più processi è stato utilizzato per ognuna di esse un semaforo di valore iniziale pari a 1.
Il processo master definisce tramite un random e successivo controllo le celle di blocco e le celle source, chiudendo il programma e stampando un messaggio se il numero è eccessivo.
Nel caso delle celle source, in base alla loro quantità possono essere:

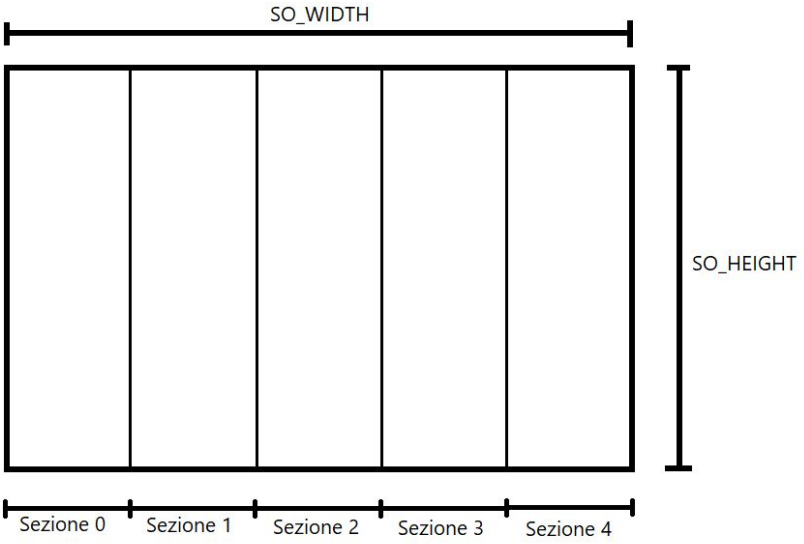
- Calcolate direttamente le celle di tipo source se la quantità piccola.
- Calcolate prima le celle che non sono source e successivamente registrate come source quelle che non sono state calcolate nella prima fase.

Generazione dei figli

Il processo master effettua una fork per ogni nuovo processo taxi e ogni nuovo processo source.
Le posizioni dei processi source sono state già calcolate e memorizzate all'interno di una array, mentre le posizioni dei taxi vengono calcolate prima di ogni loro fork.
Il processo master memorizza all'interno di due array i pid dei figli.
I nuovi processi creati dalla fork invocheranno tramite execve un nuovo file di esecuzione definendo una nuova area stack e una nuova area heap ma mantenendo il proprio PID.
Sempre tramite la execve vengono passati come argomenti la posizione del processo all'interno della mappa, variabili d'ambiente necessarie al figlio, gli id delle risorse IPC e nel caso dei taxi l'indice che rappresenta il proprio pid all'interno dell'apposito array.
Due semafori vengono utilizzati per avviare correttamente i processi figli, uno che informa il padre del completamento della loro inizializzazione e uno che permette al master di inviargli in maniera simultanea un segnale di start.

Generazione e ricezione delle richieste

Per permettere ai taxi di prendere in servizio una richiesta più possibilmente vicina a esso la mappa è stata suddivisa in **N** sezioni di dimensione D.



La dimensione della sezione e definibile nell'header.
Una funzione prendendo come parametro una posizione delle ordinate x, ne restituisce la sezione.
Per ogni sezione è stata creata una coda di messaggi, si sarebbe potuto usare una singola coda di messaggi definendo il tipo di messaggio in base alla sezione,ma tale scelta è restrittiva perché c'è il rischio che un tipo di messaggio diventi preponderante sugli altri, non permettendo l'invio di segnali da source presenti in sezioni con tipo diverso.
Definita la propria sezione di appartenenza, il processo source invierà un messaggio alla coda corrispondente alla propria sezione.
Sia nel caso in cui non ci sia spazio nella coda che nel caso in cui il messaggio sia stato inviato con successo, il processo aspetterà per un certo intervallo di nanosecondi.
A ogni coda è associato un semaforo di valore 1, che permetterà un accesso in mutua esclusione.
L'invio e la ricezione di messaggi è mascherata rispetto a segnali esterni o interni di tipo SIGALRM,SIGTERM o SIGINT, per non permettere l'interruzione involontaria di una richiesta/ricezione e impedire la non uscita del processo dalla sezione critica.
I processi taxi ricevono le richieste solo dalla coda di messaggi corrispondente alla sezione in cui si trovano, oppure,nel caso in cui siano pochi, controllano prima la coda della propria sezione e successivamente tutte quelle adiacenti.
Un processo creato dal **fork** del processo master invia ogni secondo un alarm a un processo taxi scelto caso, che gestirà il segnale da un handler modificato per inviare una nuova richiesta ai processi taxi.

Movimento taxi

Una volta ricevuta la richiesta, il taxi compie le seguenti azioni:

- Aspetta tramite nanosleep il tempo di attraversamento specificato nella cella in cui è presente.
- Sceglie la prossima casella da occupare tramite il seguente algoritmo:
    - Scelta di una direzione(sinistra,alto,destra o basso) in base all'offset tra destinazione e posizione attuale, scegliendo quando possibile una direzione diversa da quella del movimento precedente.
    - Controllo se nella direzione pressa ci siano celle inaccessibili, e in tale caso caso deviazione verso una cella accessibile.
- Tramite la funzione semtimedop() prenota la cella scelta e se non riesce ad ottenerla richiede la creazione di un nuovo taxi al master e si auto-elimina.
- Una volta ottenuta la nuova cella incrementa il semaforo della vecchia cella e aggiorna in maniera atomica (anche rispetto al print della mappa) le variabili curtaxi (taxi correnti) delle due celle.
- Aggiorna la propria posizione.

Il movimento è ripetuto finché la posizione attuale corrisponde alla destinazione. All'inizio della ricezione delle richieste viene registrato il tempo attuale, il quale verrà successivamente confrontato con il tempo rilevato all'interno del ciclo while addetto al controllo periodico delle richieste, se la differenza tra inizio della ricerca messaggi e tempo attualmente rilevato è superiore SO_TIMEOUT il processo invia al master la richiesta di creazione di un nuovo taxi e si auto-elimina.

Richiesta creazione nuovo taxi

Per gestire la richiesta di creazione di un nuovo taxi è stata creata una coda di messaggi tra processi taxi e master.
Nel richiedere la creazione di un nuovo taxi il vecchio processo taxi:

- Decrementa il variabile curtaxi in mutua esclusione ai processi concorrenti alla stessa cella e alla stampa della mappa.
- Incrementa il semaforo della cella occupata.
- Invia un messaggio al padre con l'indice che gli aveva passato durante la execve.
- Termina autonomamente o a causa del segnale di terminazione da parte del master.

Il processo master, una volta ricevuto il messaggio:

- Estrarrà dal messaggio l'indice corrispondente alla posizione all'interno dell'array di pid in cui viene memorizzato il pid processo figlio.
- Effettuerà una fork per creare un nuovo processo taxi il cui pid sostituirà quello del vecchio processo taxi nell'array.
- Effettuerà un waitpid sul vecchio pid per permettere la chiusura definitiva del vecchio processo taxi.

Stampa della mappa

Il master creerà un processo di supporto che stamperà la mappa ogni secondo tramite l'utilizzo di una nanosleep.
Per la stampa della mappa mappa viene utilizzato un semaforo di valore SO_TAXI che verrà decrementato di SO_TAXI quando il processo di supporto deciderà di stampare la mappa e incrementato di SO_TAXI quando il processo avrà finito di stampare la mappa.
Ogni qualvolta un processo decide di modificare le informazioni di una cella dovrà effettuare un wait sul semaforo per poi effettuare un signal al termine delle operazioni.

Segnale di terminazione.

Dopo aver concesso il segnale di start ai processi il master effettua una richiesta di alarm a se stesso dopo SO_DURATION secondi.
Quando verrà generato il segnale il master lo gestirà con un handler modificato.
L'handler invierà ai figli un segnale SIGTERM (inclusi i figli di support) che verrà a sua volta correttamente gestito attraverso un handler SIGTERM modificato, capace di chiudere correttamente i figli effettuando il detach della memoria condivisa e rimuovendo le varie allocazioni di memoria.
Successivamente l'handler dell'alarm aspetterà la terminazione dei processi figli creati.
Quando tutti processi figli saranno stati chiusi correttamente verranno stampati i dati statistici, eliminate le risorse IPC e rimosse aree di memoria precedentemente allocate.

Dati statistici

Ogni cella contiene una variabile "totatt" che descrive gli attraversamenti totali sulla cella, ogni qual volta un processo taxi attraversa una cella incrementa questa variabile.
Alla terminazione dell'intero programma un algoritmo individua le TOP_CELLS celle più attraversate e le stamperà su terminale.
I processi taxi registrano al loro interno il numero di richieste effettuate, il totale di celle attraversate e il viaggio completato più lungo.
Quando riceveranno il segnale di terminazione memorizzano e confronteranno i propri dati all'interno della struct principale della memoria condivisa, un semaforo sarà utilizzato per effettuare gli accessi la mutua esclusione.
Ogni volta che un processo taxi finisce di gestire una richiesta, incrementa, sempre in maniera atomica, la variabile successi all'interno della struct principale della memoria condivisa.
Analogo procedimento avviene per le variabili aborted e inevasi.