The Taxicab Game

Relazione Progetto Sistemi Operativi – a.a. 2020/21

Data: 15/01/2021

Studenti:

- ✓ Andrea Torella (Matricola 912579)
- ✓ Agostino Messina (Matricola 913813)

Sommario

Introduzione

Per facilitare e semplificare al meglio l'esecuzione del progetto, si è deciso di programmare in maniera modulare seguendo una struttura dedicata sotto riportata. Vengono anche specificate le scelte implementate.

Struttura

Master.c

Consiste nel processo principale, contiene il codice eseguibile main. Sono qui definiti i parametri a tempo di esecuzione, le funzioni indispensabili per la creazione e inizializzazione del gioco e il codice relativo ai processi source.

Taxi.c

Consiste nel codice relativo ai processi Taxi, figli del processo master, creati da questo mediante *fork* e richiamando la funzione *"create_taxi_child()"*.

Il modulo contiene tutte le funzioni relative a:

- Creazione;
- Ricerca delle richieste;
- Movimento;
- Scrittura in memoria condivisa delle statistiche relative ai viaggi effettuati;
- Terminazione.

Sem_lib.c

Si tratta del modulo contenente tutte le funzioni per la gestione e l'utilizzo dei semafori:

- Inizializzazione del semaforo;
- Estrazione della risorsa bloccante;
- Estrazione della risorsa non bloccante;
- Estrazione della risorsa con un tempo prefissato (*semtimedop*);
- Rilascio della risorsa;
- Attesa di zero;
- Inizializzazione dei semafori a un valore specificato.

Utilitiy.c

Modulo contenente:

- Macro e strutture per l'implementazione del progetto;
- Funzioni generali di utilità per lo svolgimento del programma;
- Parametri definiti a tempo di compilazione (*SO_WIDTH* e *SO_HEIGHT*);

List_lib.c

Modulo contenente:

- Struttura per la creazione di liste
- Metodi per operare sulle liste

Makefile

File per il comando Make, contenente le specifiche e i target di compilazione (*-std=c89 -pedantic*). È possibile scegliere di compilare il progetto con un determinato preset, utilizzando degli appositi comandi ("*make BUILD=dense"* o "*make BUILD=large"*).

Scelte Implementative

1.1 - Mappa della città

È stato deciso di implementare la mappa della città mediante una struttura "container" (*shared_map*) con all'interno la matrice di dimensioni *SO_WIDTH* e *SO_HEIGHT*. Questa struttura è in condivisione tra Master, Taxi e Source, ognuno dei quali effettua l'attaching.

L'inizializzazione della mappa viene effettuata dal master, mediante la funzione "*create_matrix()"*, dopo aver effettuato un controllo sui dati in input e restituire errore in caso questi non permettano di crearla.

L'algoritmo utilizzato dispone gli *SO_HOLES* in modo da non avere barriere di celle inaccessibili ed inizializza le varie celle (rappresentate dalla struttura cell) ai valori estratti casualmente tra *SO_CAP_MIN* e *SO_CAP_MAX* e *tra SO_TIMENSEC_MIN* e *SO_TIMENSEC_MAX*, oltre a inizializzare tutti gli altri parametri della cella (*crossing_time*, *crossing_cont,* ecc.).

1.2 - Richieste di servizio taxi

Queste sono generate da *SO_SOURCES* processi creati dal master. Si è scelto di non generare le richieste ogni tot secondi, ma con un intervallo variabile estratto casualmente (*random_request*). Le richieste sono generate sulla mappa senza sovrapposizione con l'utilizzo del semaforo "*id_sem_request*" e controllando che queste non vengano generate nelle celle holes. La richiesta viene quindi trasmessa sulla coda di messaggi (*id_msg_queue*) in mutua esclusione (con il semaforo "*id_sem_write*"). Fatto questo, il processo source rimane in attesa di 0 sul semaforo id_sem_request, che indica che la richiesta precedente è stata consumata e che può crearne una nuova.

La richiesta di servizio taxi può essere creata anche da terminale, inviando ad un determinato processo source un segnale *SIGUSR2* (*kill –SIGUSR2 "pid"*).

1.3 - Movimento dei taxi

I taxi vengono creati e disposti in modo casuale sulla mappa dalla funzione *create_taxi()*. Questi sono abilitati a partire solo quando tutti i taxi sono stati creati e inizializzati. Per fare questo si è scelto di usare il semaforo "*id_sem_taxi*", con valore inizializzato a *SO_TAXI + 1*. Ogni taxi, infatti, dopo essere stato creato, effettua una "*dec_sem"* e una "*wait for 0"* (stesse operazioni effettuate dal master all'inizio del gioco).

Si è scelto di implementare il meccanismo del "QUARTIERE", valore che rappresenta il numero di bordi in cui il taxi è abilitato a cercare la richiesta (Es: se QUARTIERE = 2 il taxi cerca nelle 16 celle adiacenti), in modo da far sì che questi cerchino le richieste in base alla loro convenienza. In ogni caso il taxi effettua per primo un controllo sulla sua cella di origine.

Se un taxi non trova una richiesta entro *SO_TIMEOUT* termina con la funzione "*close_taxi()"*, rilasciando le risorse utilizzate. Se stava effettuando una corsa (flag *doing_request* a 1), viene richiamata la funzione "*mutex_op(1)"* (utilizzata anche in modo *mutex_op(0)* quando la richiesta invece viene completata), la quale permette di aggiornare in mutua esclusione i vari contatori presenti all'interno della struttura "*stats*" in memoria condivisa (*strada_fatta*, *num_richieste*, *durata_viaggio*, *num_viaggi_abortiti*, ecc.).

Per evitare una situazione di stallo, i taxi utilizzano una "*dec_sem_wait"* (*semtimedop*) per l'accesso ad una cella.

Quando un processo taxi termina, il master, in attesa mediante *waitpid(),* effettua nuovamente la creazione.

1.4/1.5 - Inizio e terminazione della simulazione / Stampa

All'avvio della simulazione il master fa partire un *alarm()*, con valore *SO_DURATION*. Quando questo scade, l'handler del master imposta a 1 il flag "*stop_create"* per fare in modo che venga interrotta la "stampa ogni secondo" della capienza delle celle (effettuata da uno specifico processo figlio) e viene richiamata la funzione "*close_master()*" che effettua :

- Terminazione dei figli;
- Detaching ed eliminazione della memoria condivisa;
- Eliminazione dei semafori;
- Chiusura del processo master.