Progetto SO 2020/21
The Taxicab game

Bini, Radicioni, Schifanella

17 dicembre 2020

Indice

1 Descrizione **1**
1.1 Mappa della città 1
1.2 Richieste di servizio taxi 2
1.3 Movimento dei taxi 2
1.4 Inizio e terminazione della simulazione 2
1.5 Stampa 3

2 Configurazione **3**

3 Requisiti implementativi **4**

4 Composizione gruppo di studenti **4**

5 Consegna **4**

6 Valutazione e validità **5**

1 Descrizione

Si intende realizzare un sistema in cui sono presenti vari taxi (simulati da processi) che si spostano all'interno di una città.

1.1 Mappa della città

Le strade della città sono rappresentate da una griglia di larghezza `SO_WIDTH` e altezza `SO_HEIGHT`. La griglia ha `SO_HOLES` celle inaccessibili disposte in posizione casuale. Non ci possono essere "barriere" di celle inaccessibili: per ogni cella inaccessibile, le otto celle adiacenti non possono essere inaccessibili. Figura 1 illustra un esempio di posizionamento legale di celle inaccessibili, mentre Figura 2 mostra alcuni casi di celle inaccessibili.

Ogni cella è caratterizzata da

- un tempo di attraversamento estratto casualmente fra `SO_TIMENSEC_MIN` e `SO_TIMENSEC_MAX` (in nanosecondi)
- una capacità estratta casualmente fra `SO_CAP_MIN`, `SO_CAP_MAX` che rappresenta il numero massimo di taxi che possono essere ospitati contemporaneamente nella cella.

Le caratteristiche della cella di cui sopra sono generate casualmente all'atto della creazione della mappa e non variano per tutta la durata di una simulazione.

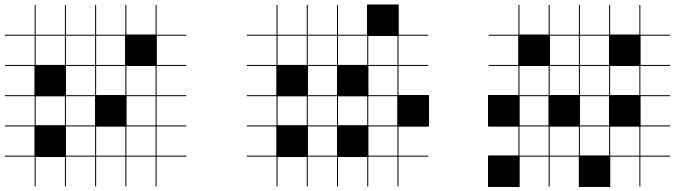

Figura 1: Esempio di posizione **consentita** delle celle inaccessibili.

1.2 Richieste di servizio taxi

Le richieste di taxi sono originate da `SO_SOURCES` processi dedicati. Ognuno di essi è legato ad una cella della mappa da cui la richiesta si genera. Ogni richiesta di servizio taxi è caratterizzata da:

- la cella di partenza (la cella fra le `SO_SOURCES` ove la richiesta è generata);
- la cella di destinazione determinata in modo casuale fra una qualunque delle celle della mappa tra quelle accessibili, diversa da quella di partenza.

Le celle `SO_SOURCES` di origine delle richieste sono posizionate casualmente sulla mappa senza sovrapposizione.

Le richieste originatesi in ognuna delle `SO_SOURCES` sorgenti sono generate da `SO_SOURCES` processi che immettono tali richieste in una coda di messaggi oppure una pipe (la scelta se avere una singola coda/pipe o una diversa per ogni sorgente è libera). I processi taxi prelevano le richieste, e le richieste possono essere servite soltanto da un taxi che si trova nella cella di origine della richiesta.

Le richieste sono create da `SO_SOURCES` processi secondo un pattern configurabile a propria scelta, per esempio attraverso un alarm periodico oppure con un intervallo variabile. Inoltre, ognuno degli `SO_SOURCES` processi deputati alla generazione di richieste deve generare una richiesta ogni volta che riceve un segnale (a scelta dello sviluppatore), per esempio da terminale.

1.3 Movimento dei taxi

All'atto della creazione i taxi sono posizionati in posizione casuale. I taxi sono abilitati alla ricerca di una richiesta da servire soltanto dopo che tutti gli altri processi taxi sono stati creati e inizializzati.

I taxi si possono muovere orizzontalmente e verticalmente, ma non in diagonale. Il tempo di attraversamento di una cella si simula con una `nanosleep(...)` di durata pari al tempo di attraversamento della cella.

Situazione di stallo Se un taxi non si muove entro `SO_TIMEOUT` secondi termina rilasciando eventuali risorse possedute. Se stava effettuando una corsa per un cliente, la richiesta viene marcata come "aborted" e quindi eliminata. Quando un processo taxi termina a causa del timeout, allora un nuovo processo taxi viene creato in una posizione casuale.

1.4 Inizio e terminazione della simulazione

La simulazione si avvia dopo che tutti i processi sorgente e processi taxi sono stati creati e inizializzati. Dal momento dell'avvio, la simulazione ha una durata `SO_DURATION` secondi che sarà un alarm per il processo master.

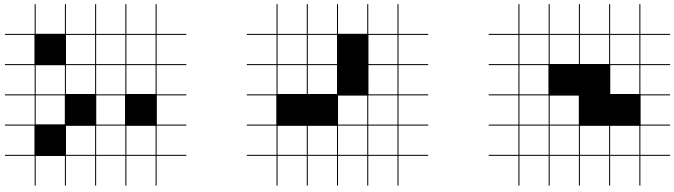

Figura 2: Esempio di posizione **non consentita** delle celle inaccessibili.

1.5 Stampa

È presente un processo "master" che raccoglie le statistiche delle varie richieste eseguite.
Ogni secondo viene stampato a terminale lo stato di occupazione delle varie celle.
Alla fine della simulazione vengono stampati:

- numero viaggi (eseguiti con successo, inevasi e abortiti)
- la mappa con evidenziate le `SO_SOURCES` sorgenti e le `SO_TOP_CELLS` celle più attraversate
- il processo taxi che
  1. ha fatto più strada (numero di celle) di tutti
  2. ha fatto il viaggio più lungo (come tempo) nel servire una richiesta
  3. ha raccolto più richieste/clienti

2 Configurazione

I seguenti parametri sono letti a **tempo di esecuzione**, da file, da variabili di ambiente, o da `stdin` (a discrezione degli studenti):

- `SO_TAXI`, numero di taxi presenti
- `SO_SOURCES`, numero di punti sulla mappa di origine dei clienti (si immagini che siano le stazioni, gli aereoporti, etc.)
- `SO_HOLES`, numero di celle della mappa inaccessibili. Si ricordi: ogni cella inaccessibile non può averne altre nelle otto celle che la circondano
- `SO_TOP_CELLS`, numero di celle maggiormente attraversate
- `SO_CAP_MIN`, `SO_CAP_MAX`, capacità minima e massima di ogni cella
- `SO_TIMENSEC_MIN`, `SO_TIMENSEC_MAX`, tempo minimo e massimo necessario per l'attraversamento di ogni cella (in nanosecondi).
- `SO_TIMEOUT`, tempo di inattività del taxi dopo il quale il taxi muore
- `SO_DURATION`, durata della simulazione

Un cambiamento dei precedenti parametri non deve determinare una nuova compilazione dei sorgenti.
Inoltre, i seguenti parametri sono letti a **tempo di compilazione**:

- `SO_WIDTH`, larghezza della mappa;
- `SO_HEIGHT`, altezza della mappa.

La Tabella 1 elenca valori "dense" e "large" per alcune configurazione di esempio da testare. Si tenga presente che il progetto deve poter funzionare anche con altri parametri fattibili. Se i parametri specificati, che possono essere scelti anche non tra quelli indicati in tabella, non permettono di generare una città valida, la simulazione deve terminare indicando la causa del problema nella generazione.

parametro	“dense”	“large”
SO_WIDTH	20	60
SO_HEIGHT	10	20
SO_HOLES	10	50
SO_TOP_CELLS	40	40
SO_SOURCES	SO_WIDTH×SO_HEIGHT−SO_HOLES	10
SO_CAP_MIN	1	3
SO_CAP_MAX	1	5
SO_TAXI	SO_SOURCES/2	1000
SO_TIMENSEC_MIN [nsec]	100000000	10000000
SO_TIMENSEC_MAX [nsec]	300000000	100000000
SO_TIMEOUT [sec]	1	3
SO_DURATION [sec]	20	20

Tabella 1: Esempio di valori di configurazione.

3 Requisiti implementativi

Il progetto deve

- essere realizzato sfruttando le tecniche di divisione in moduli del codice,
- essere compilato mediante l'utilizzo dell'utility `make`
- massimizzare il grado di concorrenza fra processi
- deallocare le risorse IPC che sono state allocate dai processi al termine del gioco
- essere compilato con almeno le seguenti opzioni di compilazione:

  ```
  gcc -std=c89 -pedantic
  ```

- poter eseguire correttamente su una macchina (virtuale o fisica) che presenta parallelismo (due o più processori).

4 Composizione gruppo di studenti

Il progetto si deve svolgere in gruppo composto al **massimo 3 da componenti**. È possibile anche svolgere il progetto da soli.

Si raccomanda che il gruppo sia composto da studenti dello **stesso turno**, i quali discuteranno con il docente del proprio turno. È consentita anche la realizzazione del progetto di laboratorio da parte di un gruppo composto da studenti di turni diversi alle seguenti condizioni. In questo caso, **tutti** gli studenti del gruppo discuteranno con **lo stesso docente**. Esempio: Tizio (turno T1) e Caio (turno T2) decidono di fare il progetto insieme. Lo consegnano e vengono convocati dal prof. Radicioni il giorno X. Tale giorno Tizio e Caio si presentano e ricevono entrambi una valutazione dal Prof. Radicioni (anche se Caio fa parte del turno T2 il cui docente di riferimento è il prof. Bini).

5 Consegna

Il progetto è costituito da:

1. il codice sorgente
2. una breve relazione che sintetizzi le scelte progettuali compiute

Il progetto si consegna compilando la seguente Google Form

- `https://forms.gle/mQGURwe89StBvB818`

la quale richiederà, oltre al caricamento del progetto stesso (un unico file in formato .zip o .tgz), anche i seguenti dati per ciascun componente del gruppo: cognome, nome, matricola, email. Dopo il caricamento del progetto, verrete convocati dal docente con cui discuterere (si veda Sezione 4 in caso di gruppo composto da studenti di turni diversi). Attenzione: **consegnare il progetto una volta sola**. Una eventuale ulteriore consegna prima dell'appuntamento **annullerà la data dell'appuntamento**.

La consegna deve avvenire almeno **10 giorni prima** degli appelli scritti per dare modo al docente di pianificare i colloqui:

- se spedite 10 giorni prima di un appello, il docente propone una data per la discussione entro l'appello seguente
- altrimenti, la data sarà dopo l'appello seguente.

6 Valutazione e validità

Il progetto descritto nel presente documento potrà essere discusso inviando l'email al docente entro Novembre 2021. Da Dicembre 2021 sarà discusso il progetto assegnato durante l'anno accademico 2021/22.

Tutti i membri del gruppo devono essere presenti alla discussione. La valutazione del progetto è **individuale** ed espressa in 30-esimi. Durante la discussione

- verrà chiesto di illustrare il progetto
- verranno proposti quesiti sul programma "Unix" del corso

È necessario ottenere una votazione di almeno **18** su 30 per poter essere ammessi allo scritto. In caso di superamento della discussione del progetto, la votazione conseguita consentirà di partecipare allo scritto per i **cinque appelli successivi** alla data di superamento.

In caso di mancato superamento, lo studente si potrà ripresentare soltanto dopo almeno **un mese** dalla data del mancato superamento

Si ricorda che il voto del progetto ha un peso di $\frac{1}{4}$ sulla votazione finale di Sistemi Operativi.