di.unito.it
DIPARTIMENTO DI INFORMATICA
Università degli Studi di Torino

The Taxicab game

Progetto Sistemi Operativi
a.a 2020/2021

Ramiro Calò matr. 835678
Carla Claudia Pricop matr. 849972
Dominique Bisanti matr. 836318

Indice

1. Introduzione

Per verificare le competenze di laboratorio apprese dai propri studenti del corso di Sistemi Operativi i professori hanno assegnato un progetto da sviluppare e consegnare chiamato "The Taxicab game".
In questo documento sono sintetizzate le scelte progettuali compiute.

1.1. Obiettivo

L'obiettivo del progetto è quello di realizzare realizzare un sistema in cui sono presenti vari taxi (simulati da processi) che si spostano all'interno di una città.

Il gioco si svolge in round gestiti da un processo Master.

2. Requisiti e vincoli allo sviluppo

Il progetto è stato sviluppato, come richiesto, utilizzando le tecniche di divisione in moduli del codice.

E' stata utilizzata l'utility make.

E' stato massimizzato il grado di concorrenza fra processi.

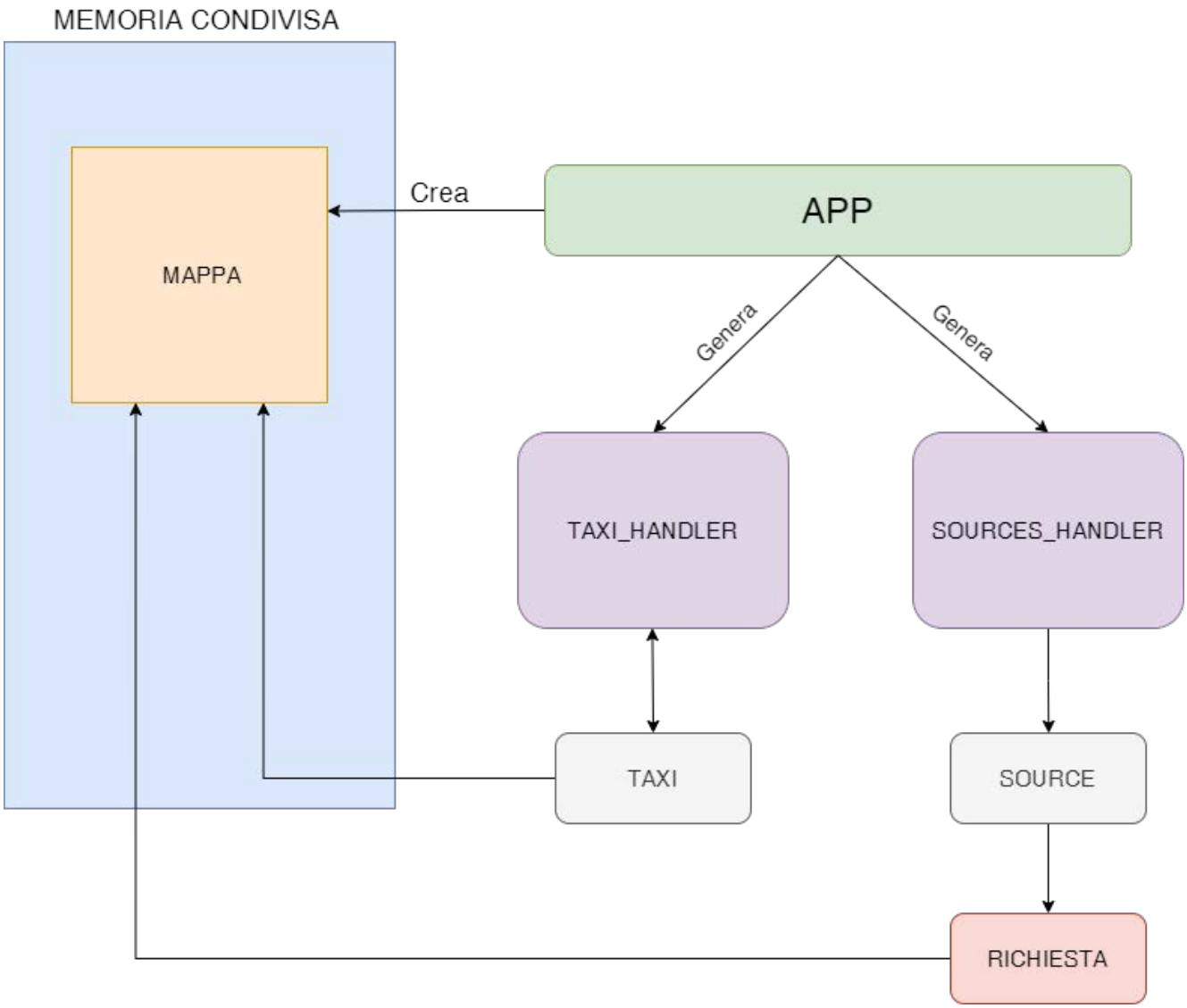
Il makefile contiene l'opzione di compilazione **gcc -std=c89 -pedantic.**

Al termine della simulazione tutte le risorse IPC che sono state allocate dai processi vengono deallocate.

Viene eseguito correttamente su una macchina virtuale o fisica che presenta parallelismo(2 o più processori).

3. Descrizione della soluzione software

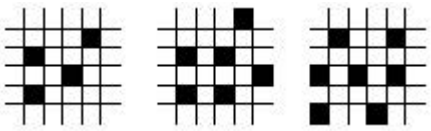
3.1. Diagramma generale



3.2. Mappa della città

Per la generazione della mappa abbiamo deciso di utilizzare una matrice in memoria condivisa di dimensione **SO_WIDTH*SO_HEIGHT**.

La matrice presenta delle celle inaccessibili chiamate **SO_HOLES** disposti in posizioni casuali, ogni **SO_HOLES** ha le 8 celle adiacenti inaccessibili.



Ogni cella è una struct caratterizzata da una:

- one : intero per definizione hole;
- soCapMax: capacità massima della cella;
- soCap: capacità corrente della cella;
- soTime: tempo di attraversamento cella;
- source: intero per definizione source;
- count: contatore di attraversamento taxi;

3.3. Configurazioni

Per evitare inutili compilazioni quando si cambiano i parametri di simulazione abbiamo creato un file conf.csv che viene parsificato dalla funzione **parseConf (conf* confg, char * percorso)** attraverso un puntatore di tipo conf, che è una struttura creata per contenere i dati di settaggio.

In questo modo non è necessario ricompilare il codice se si effettua una modifica ai parametri di configurazione perché viene letta in fase di esecuzione.

3.4. APP

Il processo APP,legge il file di configurazione, crea la mappa in memoria condivisa tramite la funzione **shmget(..)**, posiziona gli **SO_HOLES** all'interno della mappa, genera **TAXI_HANDLER** e **SOURCES_HANDLER**.
APP imposta un timer di durata **SO_DURATION** tramite la funzione **alarm(..)**.
APP attende la terminazione dei processi figli.

3.5. TAXI_HANDLER

Il processo **TAXI_HANDLER** si occupa di generare **SO_TAXI** e di gestire la loro terminazione attraverso la generazione di un nuovo **TAXI**.

3.6. SOURCES_HANDLER

Il processo **SOURCES_HANDLER** si occupa di generare **SO_SOURCES** e di gestire la loro terminazione.

3.7. TAXI

Il processo TAXI esegue un attached alla memoria condivisa, si posiziona casualmente all'interno della mappa, setta un timer di esecuzione con la funzione **alarm(..)** e legge dalla coda di messaggi per acquisire una richiesta. Il processo TAXI esegue il movimento tramite la funzione movimentoManhattanSEC(..) e scrive su una seconda coda di messaggi il fallimento o il successo della richiesta.

Scaduto il timer termina.

3.8. SOURCE

Il processo SOURCE esegue un attached alla memoria condivisa e si posiziona casualmente all'interno della mappa.

Attraverso la chiamata di nanosleep(..) ogni secondo il processo esegue una chiamata alla funzione raise(..) con la quale genera un nuovo processo richiesta.

3.9. RICHIESTA

Il processo RICHIESTA scrive un messaggio nella coda di messaggi principale attraverso cui si identifica con il proprio pid e comunica la sua posizione all'interno della mappa e il punto nella mappa di arrivo.

Il processo si mette in attesa di ricevere un messaggio nella coda di messaggi secondaria, il messaggio contiene l'esito della richiesta(fallimento/successo). Tale esito viene registrato nelle statistiche.

3.10. LIBRERIE

- conf.c : libreria per il parsing;
- key.c : libreria gestione chiavi;
- manhattan.c : libreria movimento taxi;
- shmLibs.c : libreria improntata sulla gestione della memoria condivisa;
- sigLib.c : libreria handler segnale;

4. Stampa