RELAZIONE PROGETTO DI SISTEMI OPERATIVI 2020-2021

Tuninetti André matr. 864115
Catalin Marian Sandur matr. 862329

In questa simulazione per rappresentare la città abbiamo scelto di utilizzare un array bidimensionale contenente delle celle. Queste sono state implementate con una struct e contengono al loro interno l'id, le coordinate, la capacità, il tipo di cella ed altri dati utili.

La matrice di celle descritta sopra è inserita a sua volta in una struct ed allocata in memoria condivisa insieme ai valori necessari a far terminare nei tempi previsti la simulazione. Per distinguere i tre tipi di cella usiamo la variabile type che indica un buco nel caso in cui valga 0, una cella normale nel caso valga 1 e una cella di tipo SO_SOURCE quando vale 2.

Abbiamo poi implementato un semaforo all'interno di ogni cella per evitare che quest'ultima accolga più taxi di quanti ne possa ospitare contemporaneamente in base alla sua capacità.

I processi SO_SOURCES vengono generati ed attendono che tutti gli altri processi siano stati inizializzati. A questo punto iniziano a generare richieste e le scrivono sulla message queue.

I processi taxi vengono generati in modo casuale e dopo aver atteso su un semaforo wait for zero che tutti gli altri processi siano stati inizializzati, come prima mossa, si recano verso una cella di tipo source. Una volta raggiunta quella iniziano a leggere i messaggi che nel frattempo sono stati inviati dai processi SO_SOURCES nella message queue. In questo caso sfruttiamo il campo mtype del messaggio per ricevere solo richieste provenienti dalla cella in cui il taxi si trova al momento della lettura. Una volta letto il messaggio il taxi ottiene la sua nuova destinazione e cerca di raggiungerla con la funzione drive().

All'interno di questa funzione calcoliamo la distanza di manhattan per scegliere verso quale cella sia più conveniente spostarsi al fine di raggiungere la destinazione.
L'accesso alle celle è regolamentato dai loro semafori che vengono inizializzati con un valore uguale alla loro capacità. Avviene anche un controllo per impedire che la cella scelta sia un buco.
Qualora un processo rimanga in attesa per più di TAXI_TIMEOUT secondi termina con EXITSTATUS = 129. Nel processo padre questo viene ricevuto e determina la creazione di un nuovo taxi che prenderà il posto di quello appena terminato.

Allo scadere di SO_TIMEOUT viene generato un alarm gestito dal signal_handler tramite il quale notifichiamo i processi SO_TAXI e SO_SOURCES in esecuzione con un SIGUSR1. Quando un processo riceve questo segnale termina.

Al termine della simulazione il processo padre recupera dal segmento di shared memory le statistiche da stampare, disalloca le strutture addette alla comunicazione tra processi e termina a sua volta.