Machine Translated by Google

SO Project 2020/21
The Taxicab game

Bini, Radicioni, Schifanella

December 18, 2020

Index

1 Description

We intend to create a system in which there are various taxis (simulated by processes) that move within a city.

1.1 City map

The city streets are represented by a grid of width SO_WIDTH and height SO_HEIGHT. The grill has SO_HOLES inaccessible cells arranged in random position. There can be no “barriers” of inaccessible cells: for each inaccessible cell, the eight adjacent cells cannot be inaccessible. Figure 1 illustrates an example of legal placement of inaccessible cells, while Figure 2 shows some cases of inaccessible cells.

Each cell is characterized by

- a crossing time randomly drawn between SO_TIMENSEC_MIN and SO_TIMENSEC_MAX (in nanoseconds)
- a capacity randomly extracted between SO_CAP_MIN, SO_CAP_MAX which represents the maximum number of taxis which can be housed simultaneously in the cell.

The characteristics of the cell above are randomly generated when the map is created and not vary throughout the duration of a simulation.

Machine Translated by Google

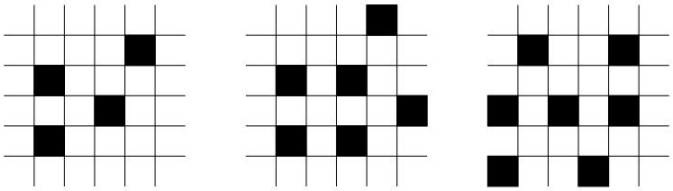

Figure 1: Example of allowed location of inaccessible cells.

1.2 Requests for taxi service

Taxi requests originate from SO_SOURCES dedicated processes. Each of them is linked to a cell of the map from which the request is generated. Each taxi service request is characterized by:

- the starting cell (the cell among the SO_SOURCES where the request is generated);
- the destination cell determined randomly from any of the cells of the map between those accessible, different from the starting one.

The request source SO_SOURCES cells are randomly placed on the map without overlap.

The requests originating in each of the SO_SOURCES sources are generated by SO_SOURCES processes that insert these requests into a message queue or a pipe (the choice whether to have a single queue/pipe or a different one for each source is free). Taxi processes fetch requests, and requests can only be served by a taxi that is in the request's origin cell.

The requests are created by SO_SOURCES processes according to a configurable pattern of your choice, for example through a periodic alarm or with a variable interval. Furthermore, each of the SO_SOURCES processes responsible for generating requests must generate a request every time it receives a signal (at the developer's choice), for example from a terminal.

1.3 Movement of taxis

Upon creation the taxis are placed in random positions. Taxis are enabled to look for a request to serve only after all other taxi processes have been created and initialized.

Taxis can move horizontally and vertically, but not diagonally. The crossing time of a cell is simulated with a nanosleep(...) lasting equal to the cell crossing time.

Deadlock If a taxi does not move within seconds SO_TIMEOUT ends by releasing any resources it has. If you were making a trip for a customer, the request is marked as “aborted” and then deleted. When a taxi process terminates due to timeout, then a new taxi process is created at a random location.

1.4 Starting and ending the simulation

The simulation starts after all source processes and taxi processes have been created and initialized. From the moment of startup, the simulation has a duration of SO_DURATION seconds which will be an alarm for the master process.

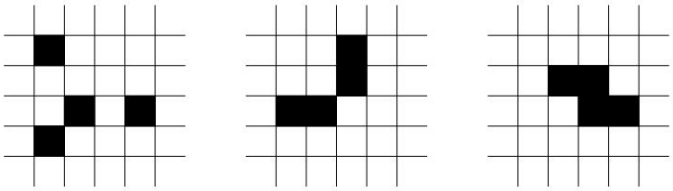

Figure 2: Example of prohibited position of inaccessible cells.

Machine Translated by Google

1.5 Print

There is a "master" process that collects the statistics of the various requests performed.
Every second the occupancy status of the various cells is printed on the terminal.
At the end of the simulation the following are printed:

- number of trips (successfully completed, unfulfilled and aborted)
- the map with the source SO_SOURCES and the most crossed SO_TOP_CELLS cells highlighted
- the taxi process that
    1. he traveled further (number of cells) than everyone
    2. he traveled the longest (in terms of time) in serving a request 3. he collected more requests/customers

2 Configuration

The following parameters are read at runtime, from file, from environment variables, or from stdin (at student discretion):

- SO_TAXI, number of taxis present
- SO_SOURCES, number of points on the customer origin map (imagine that they are stations, airports, etc.)
- SO_HOLES, number of inaccessible map cells. Remember: each inaccessible cell cannot have others in the eight cells that surround it
- SO_TOP_CELLS, number of cells most crossed
- SO_CAP_MIN, SO_CAP_MAX, minimum and maximum capacity of each cell
- SO_TIMENSEC_MIN, SO_TIMENSEC_MAX, minimum and maximum time needed to cross each cell (in nanoseconds).
- SO_TIMEOUT, taxi inactivity time after which the taxi dies
- SO_DURATION, simulation duration

A change of the previous parameters must not lead to a new compilation of the sources.
Additionally, the following parameters are read at compile time:

- SO_WIDTH, map width;
- SO_HEIGHT, height of the map.

Table 1 lists “dense” and “large” values for some example configurations to test. Please note that the project must also be able to work with other feasible parameters. If the specified parameters, which can be chosen even if not among those indicated in the table, do not allow generating a valid city, the simulation must end by indicating the cause of the problem in the generation.

Machine Translated by Google

parameter	“dense”	“large”
SO_WIDTH	20	60
SO_HEIGHT	10	20
SO_HOLES	10	50
SO_TOP_CELLS	40	40
SO_SOURCES	SO_WIDTH×SO_HEIGHTÿSO_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

Table 1: Example of configuration values.

3 Implementation requirements

The project must

- be created by exploiting techniques for dividing the code into modules,
- be compiled using the make utility
- maximize the degree of competition between processes
- deallocate the IPC resources that were allocated by the processes at the end of the game
- be compiled with at least the following compilation options:

  gcc -std=c89 -pedantic

- be able to execute correctly on a machine (virtual or physical) that presents parallelism (two or more processes)
  sori).

4 Composition of the group of students

The project must be carried out in a group composed of a maximum of 3 members. It is also possible to carry out the project yourself.

It is recommended that the group is made up of students from the same shift, who will discuss with the teacher
own turn. The realization of the laboratory project by a group composed of students from different shifts is also permitted under the following
conditions. In this case, all students in the group will discuss with him
same teacher. Example: So-and-so (turn T1) and So-and-so (turn T2) decide to do the project together. They deliver it
and are summoned by the prof. Radicioni on day X. On that day Tizio and Caio show up and both receive a
evaluation by Prof. Radicioni (even if Caio is part of the T2 shift whose reference teacher is Prof. Bini).

5 Delivery

The project consists of:

1. the source code
2. a short report summarizing the design choices made

The project is delivered by filling out the following Google Form

Machine Translated by Google

- https://forms.gle/mQGURwe89StBvB818

which will require, in addition to uploading the project itself (a single file in .zip or .tgz format), also the following data for each member of the group: surname, first name, serial number, email. After uploading the project, you will be called by the teacher to discuss (see Section 4 in case of a group made up of students from different shifts).
Please note: submit the project only once. Any further delivery before the appointment will cancel the appointment date.

The delivery must take place at least 10 days before the written exams to give the teacher the opportunity to plan the interviews:

- if sent 10 days before an exam, the teacher proposes a date for the discussion within the following exam
- otherwise, the date will be after the next appeal.

6 Evaluation and validity

The project described in this document can be discussed by sending the email to the teacher by November 2021.
From December 2021 the project assigned during the 2021/22 academic year will be discussed.

All group members must be present at the discussion. The evaluation of the project is individual and expressed in 30-ths. During the discussion

- you will be asked to explain the project
- questions will be asked about the “Unix” program of the course

It is necessary to obtain a grade of at least 18 out of 30 to be admitted to the writing. If the project discussion is passed, the score obtained will allow you to participate in the written exam for the five exam sessions following the passing date.

In case of failure to pass, the student will be able to reapply only after at least one month from the date of failure to pass

Please remember that the project vote has a weight of $\frac{1}{4}$ on the final vote of Operating Systems.