PPC Project

At the crossroads

Razmig Kéchichian & Tristan Roussillon

1. Goal

The goal of this programming project is to design and implement a multi-process simulation in Python. This document defines the functionalities that you must implement at a minimum. Some questions are deliberately left open for you to propose your own solution. Any extensions and developments will act in your favor in the evaluation. Be rigorous and creative!

2. Presentation

2.1 User view

Let's consider a crossroads made up of the perpendicular intersection of 2 roads, one running North-South, the other West-East. This intersection is managed by 4 bicolor lights, one at each corner. In normal conditions, lights on one road are the same color at all times and the opposite color on the perpendicular road, i.e. when it's red on one road, it's green on the other and vice versa (orange will not be considered). Vehicles respect traffic regulations: they proceed only on green light, have priority in the intersection when they want to turn right. If a high-priority vehicle (firefighter, ambulance, etc.) arrives on any road, it must be able to pass as quickly as possible. To achieve this, as soon as the vehicle approaches the intersection, it is detected and the lights change color to enable it to pass in the desired direction. In this case, only one of the 4 lights is set to green. We assume that no vehicle gets stuck in the middle of the intersection, i.e. there are no traffic jams.

2.2 Technical specifications

Your implementation should involve at least 5 processes:

- `normal_traffic_gen`: simulates the generation of normal traffic. For each generated vehicle, it chooses source and destination road sections randomly or according to some predefined criteria.
- `priority_traffic_gen`: simulates the generation of high-priority traffic. For each generated vehicle, it chooses source and destination road sections randomly or according to some predefined criteria.
- `coordinator`: allows all vehicles (priority or not) to pass according to traffic regulations and the state of traffic lights.
- `lights`: changes the color of the lights at regular intervals in normal mode, it is notified by `priority_traffic_gen` to set the lights to the appropriate color.
- `display`: allows the operator to observe the simulation in real-time.

Inter-process communication: The 4 sections of the crossroads are represented by message queues, one per section, vehicles are represented via messages coding the vehicle’s attributes. The approach of a high-priority vehicle is notified to the `lights` process by a signal. The state of traffic lights is stored in a shared memory, accessible to the `coordinator` processes, at least. Communication with the `display` process is carried out via sockets.

3. Implementation

3.1. Design

Start with a diagram to better visualize the interactions between processes/threads. State machine diagrams can be very helpful during this stage. The following points need to be addressed and justified:

- relationships between processes (parent-child or unrelated), define a parent-child relationship only if absolutely necessary,
- message queues used in communication and messages exchanged between processes along with their types, content and order of exchange,
- sockets used in communication between processes along with exchanged data, their content and order of exchange,
- data structures stored in shared memory, their types and how they are accessed,
- signals exchanged between processes along with their types and associated handlers,
- synchronization primitives to protect access to shared resources or to count resources,
- tubes involved in pairwise process communication, if any, and their types.

Write down a Python-like pseudo-code of main algorithms and data structures for each process/thread to help you in implementation.

3.2. Implementation plan

Plan your implementation and test steps. We strongly encourage you to first implement and test every process/thread separately on hard-coded data in the beginning. Then you can implement and test each pair of communicating processes/threads, again on hard-coded data in the beginning, if necessary. Only after implementing and testing each inter-process communication you can put all processes together and test the simulation. Take care of the startup and the proper shutdown of the simulation, freeing all resources. Identify possible failure scenarios and think about recovery or termination strategies. Lastly, make sure you are able to demonstrate your implementation during a short presentation.

4. Deadlines

09/01/2025	project is published on Moodle
06/02/2025 - 23:59	submission of project code (including a README explaining how to execute it) and report archive on Moodle, refer to organization slides for the content of the report
07/02/2025	15-minute demonstration per project with the tutor in charge of your group