# Report Lab 11: priority queue

Aurora Gensale s303535

## 1 Problem statement

In this lab, we were requested to replicate a priority queue, which means we had two sorts of customers based on their priority: high or low, with multiple servers and a waiting line of a predetermined size.
High priority clients must be served before low priority customers, and if all servers are busy, one must be kept idle to serve the high priority client. The low priority one, whose service has been terminated, is reinserted into the queue if the maximum size is not reached; otherwise, it is deleted.

## 2 Input parameters

- k = 2 (number of servers)
- N = 1000 (maximum length of the waiting line)
- Arrival processes for both high and low priority are Poisson processes with $\lambda_{HP} = \lambda_{LP} = [0.2, 0.4, 0.8, 1.4, 2.0, 2.4, 2.8]$
- Considering service times we explore two scenarios:
  a) $E[S]_{HP} = E[S]_{LP} = 1.0$
  b) $E[S]_{HP} = \frac{1}{2}$ and $E[S]_{LP} = \frac{3}{2}$
- Moreover service times are distributed in three different ways:
  - exponentially distributed with mean $E[S]_{*P}$
  - deterministic $= E[S]_{*P}$
  - distributed as an hyper-exponential random variable with mean $= E[S]_{*P}$ and standard deviation $= 10E[S]_{*P}$
- Maximum simulation time is set = 10000 for deterministic and exponential service times distributions, while for the hyper-exponential one is equal to 1000 due to high computational cost needed.

## 3 Main algorithms for simulation

The simulation is created to explore all possible combinations of

$$\lambda_{*P} \text{ x } E[S]_{*P} \text{ x service time distribution.}$$

The primary structure is fairly similar to those developed in previous labs where we had to implement queuing systems (lab 1 and 9). The main distinction is that we must manage more than one server and two different client categories here.
At the initial time = 0, the first event added to the Future Event Set (FES) has a low priority. In order to serve first the high priority clients, FES is sorted according priority; **note** that in python ("HP" ¡ "LP") = True.
Then, when an *arrival* event occurs, we determine client priority at random in order to avoid favoring one type over another and we select one server, if idle, make it busy.
When we have all servers busy and we need o start serving and high priority customer, we randomly stop one of those servers, put the customer stopped again in the waiting line, if there is still room (**recall**: waiting line has a fixed length) and make the server busy with the high priority customer.
Departure events are managed like in the previous labs, we pop one client from the queue, generate its service time and update the FES with this *departure* event. Here also delay time, interpreted as the time spent in the waiting line, is computed; **note:** in the computation we do not consider service time.
Simulation stops when maximum time is reached, outputs are plots in which we compare delays for the priorities, figure 1 shows an example.

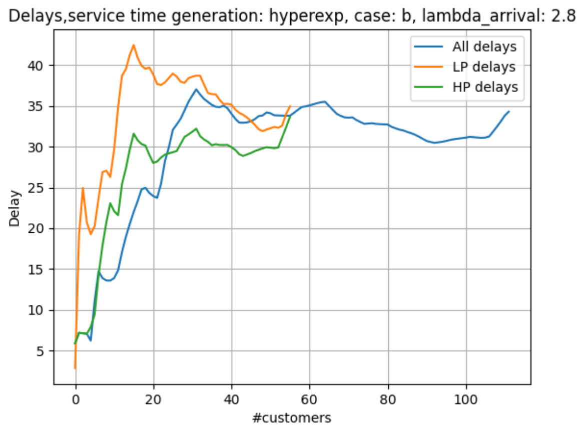

Figure 1: Average cumulative delay in case b) with $\lambda_{HP} = \lambda_{LP} = 2.8$. From this we can notice that we have a higher delay for the low priority customers.