Mini-project SNI 19–20

1/ Introduction. As seen during the course, a basic tool when analyzing a complex system such a video distribution network is a queueing model, taken in a wide sense (single-server queues, infinite-server queues, networks of queues such as tandems, etc.). This is for instance illustrated by Srikant's paper on the performances of a BitTorrent-like version of these networks. From a general point of view, simulation is the mandatory tool for these performance evaluation studies (an exception is Srikant's text where the model leads easily to simple differential equations). In the course, we saw a few examples of simulation codes for studying these types of models. This short simulation-based mini-project will consist in other analysis of these fundamental models.

2/ Code to work from. The project basically means building simple variants of the two main examples seen in the lectures, which are a simulator for the $M/M/1$ model and a second one for the $M/M/\infty$ one. You have the `C` versions of these programs and the `python` ones.

3/ Organization. Basically, you have to write variants of these two models as explained below in point 4/, and run the simulators in order to explore specific issues suggested in that point of this mini-project description. Then, you prepare a short set of slides to make a presentation of your work (targeting around 20 minutes), focusing on the analysis done and the results obtained, and you present them to us (and to your colleagues) in January.

Before the day of the presentations, you send us the used codes, the related files, some "readme" file(s), and a copy of your slides.

As said in the mails, organize yourselves in groups of 3 students, in order to put 4 presentations in a slot of two hours. For the talk, you share the presentation between the 3 members of the group, 7 or 8 minutes each. We will use some minutes for a few questions. Avoid having too many slides in order to control the lengths of the talks.

4/ Tasks.

1. The examples given during the course use as a stopping rule the total simulation time T. Modify it adding a last parameter K equal to the total number of customers *served*. For instance, for the $M/M/1$ queue, `C` version, you could have a command line call

```
./MM1  mtba  mst  T  seed  K
```

The program must stop when K customers have been served. So, leave the T parameter and the main loop as it is, comparing the date of next event to happen with this total time T, as a safeguard, and add a counter that you compare with K to stop the program. For the comparisons, you can use $K = 1000$ for your verifications, then $K = 10000$ for the results[1]. You can try higher values of K as well, of course. If T is too small so that less than K customers are served, just increase it.

We keep as a first goal to evaluate the mean number of customers $\mathbb{E}(N_\infty)$ in the model, and for simplicity, we don't add an evaluation of a confidence interval for the metric.

2. First of all, make a simulator for the $M/D/1$ queue (constant –i.e. deterministic – service time) based on your new $M/M/1$ code. For that, you must touch just a few lines, exactly at the places where you use the service time; for the $M/D/1$, instead of sampling it, you just use a constant.

   Then, the goal is to compare the metric $\mathbb{E}(N_\infty)$ in the two models, both with the same arrival processes and with the same mean service times. Using the *load* ϱ as the parameter for the comparisons, plot a curve giving $\mathbb{E}(N_\infty)$ for different values of ϱ (example: $\varrho = 0.1, 0.2, \ldots, 0.9$) in the two models. For instance, since $\varrho =$ `mst/mtba`, fixing `mtba` to 1, we have $\varrho =$ `mst`, and an example of comparison could be the two calling lines

   ```
   ./MM1  1.0  0.8  1000000  12345  10000
   ./MD1  1.0  0.8  1000000  987654321  10000
   ```

   corresponding to $\varrho = 0.8$. If you are using `python`, this could look

   ```
   python3  ./MM1.py  1.0  0.8  1000000  12345  10000
   python3  ./MD1.py  1.0  0.8  1000000  987654321  10000
   ```

3. Repeat previous task but now comparing the $M/M/\infty$ and the $M/D/\infty$ models. In this case, ϱ needs not to be less than 1, so, take something such as $\{0.1, 0.5, 1.0, 2.0, 5.0, 10.0\}$ to vary the load in.

4. The two last points are a bit more interesting, specially when dealing with video streaming applications, for example. First, here at 4/, you must add to the basic codes used in Task 2, some lines in order to be able to measure also the mean *delay* or mean *response time* (say $\mathbb{E}(R_\infty)$) as described below.

   You must add a list working as a FIFO *queue data structure* that mimics the content of the buffer at all times. The list starts empty, and each time a customer

[1]As a rule of thumb in simulation, we say that we need at least a few hundreds of events of interest to start to get a "reasonable" accuracy. This is why 10000 served customers looks ok.

arrives, we add an element at its bottom, with as data only the customer's arrival time. When you handle a departure in the main loop, you are at the (virtual) time `time`, so you remove from the auxiliary list its *head*, you evaluate `time` minus the arrival time of that unit, and you have the response time of the leaving customer. Your output will then be, for the response time of the model, the average of the individual delays for the K first customers to leave the system.

5. When we use a streaming video system, the delay in receiving the IP packets is, in general, not important for the Quality of Experience (of course, if it is not too large, or if we are looking at some sport live diffusion, etc.). The important thing is the *variation* of the delays, generally called the *jitter* of the system. This can be quantified in different ways. Let us consider here the metric $J_n = |R_n - R_{n-1}|$ for packet $n \geq 2$, where R_k is the response time of customer k.

   Repeat the Task 2 now, but evaluating the average $(J_2 + J_3 + \cdots + J_K)/(K-1)$ (seen as an estimation of the corresponding theoretical metric $\mathbb{E}(J_\infty)$) as a function of the load, in both the $M/M/1$ and the $M/D/1$ models.

Important final remarks. If you prefer using a different language, go ahead, no problem, but you will have to implement the basic discrete event simulation tools, from the `C` or the `python` versions.

If you implement things differently, for instance, making a single code for Task 2 that performs all the simulations for the considered values of ϱ and for the two models, everything in a single program, that's ok as well.

Option: instead of a single run as in the call "`./MM1 1.0 0.8 T 12345 10000`" to estimate $\mathbb{E}(N_\infty)$ or $\mathbb{E}(R_\infty)$ or $\mathbb{E}(J_\infty)$, you can do, say, 10 or 20 runs with different seeds, and use as the final result the average of the corresponding 10 or 20 outputs (this is the kind of thing to do for building a confidence interval).