Magistrello Camilla (4512554)

Assignment 1: scheduling simulator

In this assignment I had implemented a discrete event simulation.

The inputs of the simulation are:

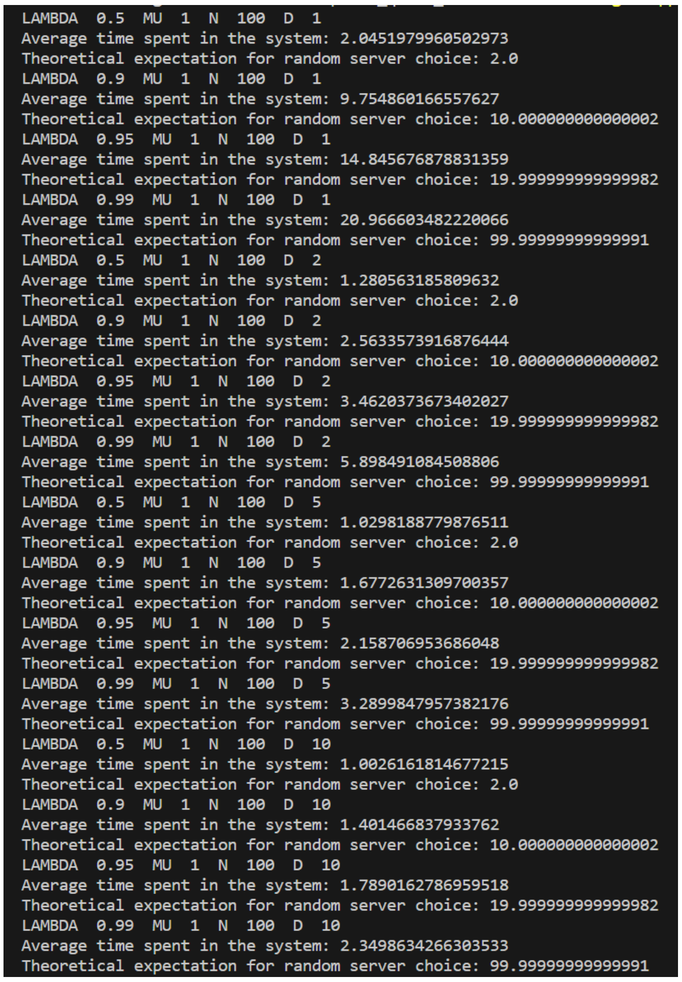
- λ: average frequency (rete) at which jobs join the queue
- μ: average frequency (rete) at which jobs leave the system when they are complete
- n: number of servers in the system
- d: sub number of server

I test the simulation with 100 of servers, μ = 1 and:

- λ = [0.5, 0.9, 0.95, 0.99]
- d = [1, 2, 5, 10]

For each of the simulations we calculated the average time spent in the system for every job and the theoretical expectation for random server choice.

These are the results I obtained:



```
LAMBDA  0.5  MU  1  N  100  D  1
Average time spent in the system: 2.0451979960502973
Theoretical expectation for random server choice: 2.0
LAMBDA  0.9  MU  1  N  100  D  1
Average time spent in the system: 9.754860166557627
Theoretical expectation for random server choice: 10.000000000000002
LAMBDA  0.95  MU  1  N  100  D  1
Average time spent in the system: 14.845676878831359
Theoretical expectation for random server choice: 19.999999999999982
LAMBDA  0.99  MU  1  N  100  D  1
Average time spent in the system: 20.966603482220066
Theoretical expectation for random server choice: 99.99999999999991
LAMBDA  0.5  MU  1  N  100  D  2
Average time spent in the system: 1.280563185809632
Theoretical expectation for random server choice: 2.0
LAMBDA  0.9  MU  1  N  100  D  2
Average time spent in the system: 2.5633573916876444
Theoretical expectation for random server choice: 10.000000000000002
LAMBDA  0.95  MU  1  N  100  D  2
Average time spent in the system: 3.462037367340202
Theoretical expectation for random server choice: 19.999999999999982
LAMBDA  0.99  MU  1  N  100  D  2
Average time spent in the system: 5.898491084508806
Theoretical expectation for random server choice: 99.99999999999991
LAMBDA  0.5  MU  1  N  100  D  5
Average time spent in the system: 1.0298188779876511
Theoretical expectation for random server choice: 2.0
LAMBDA  0.9  MU  1  N  100  D  5
Average time spent in the system: 1.6772631309700357
Theoretical expectation for random server choice: 10.000000000000002
LAMBDA  0.95  MU  1  N  100  D  5
Average time spent in the system: 2.158706953686048
Theoretical expectation for random server choice: 19.999999999999982
LAMBDA  0.99  MU  1  N  100  D  5
Average time spent in the system: 3.2899847957382176
Theoretical expectation for random server choice: 99.99999999999991
LAMBDA  0.5  MU  1  N  100  D  10
Average time spent in the system: 1.0026161814677215
Theoretical expectation for random server choice: 2.0
LAMBDA  0.9  MU  1  N  100  D  10
Average time spent in the system: 1.401466837933762
Theoretical expectation for random server choice: 10.000000000000002
LAMBDA  0.95  MU  1  N  100  D  10
Average time spent in the system: 1.7890162786959518
Theoretical expectation for random server choice: 19.999999999999982
LAMBDA  0.99  MU  1  N  100  D  10
Average time spent in the system: 2.3498634266303533
Theoretical expectation for random server choice: 99.99999999999991
```

It can be observed that as the size of the subset of d servers increases, the average time a job spends in a queue decreases. This is because I can choose the shortest queue to place the job, allowing its completion to be scheduled more quickly.

To calculate the average queue length (with n = 100), I took measurements of the queues each time a job with an even ID entered the queue of a server, and saved them in CSV files named {d}_{λ}.csv.

Since the size of these files increases with a higher number of servers in the simulation, if the total number of servers is increased, measurements can be taken, for example, when the job ID is a multiple of a larger number, to limit the overall file size.

For instance, I also tested the simulation with 200 servers, measuring the queue lengths each time the job ID was a multiple of 4. The simulation was slightly slower than the one with 100 servers, but the obtained results are consistent with those observed in the theory.

The results obtained with 200 servers, measuring the queues when the job ID is a multiple of 4, closely resemble those obtained with 100 servers, measuring the queues when the job ID is even:

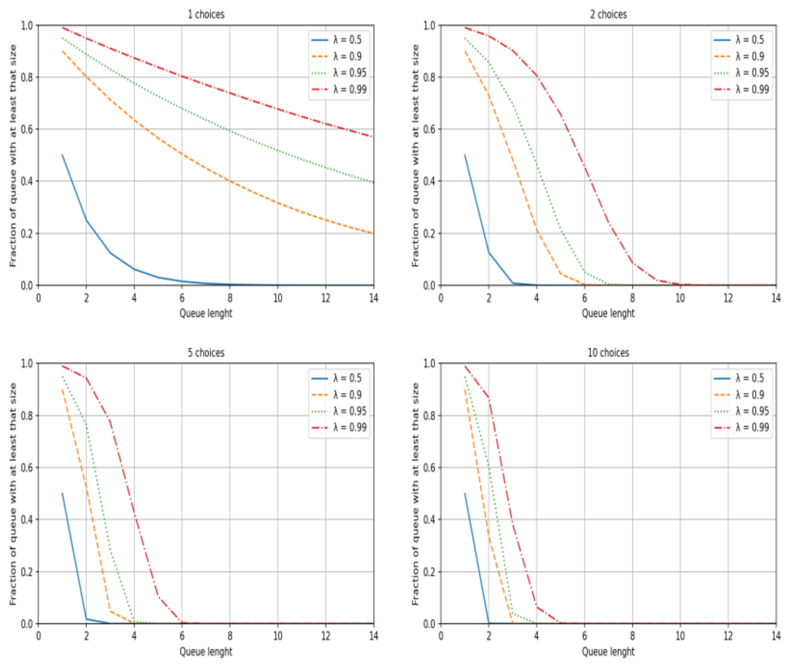


Figure 1: Results with n = 200 and measurements taken when the job ID is a multiple of 4.

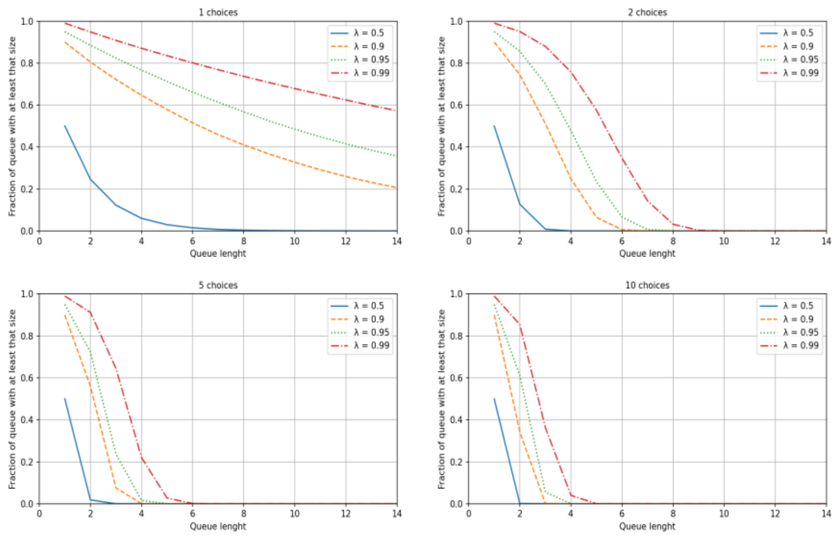
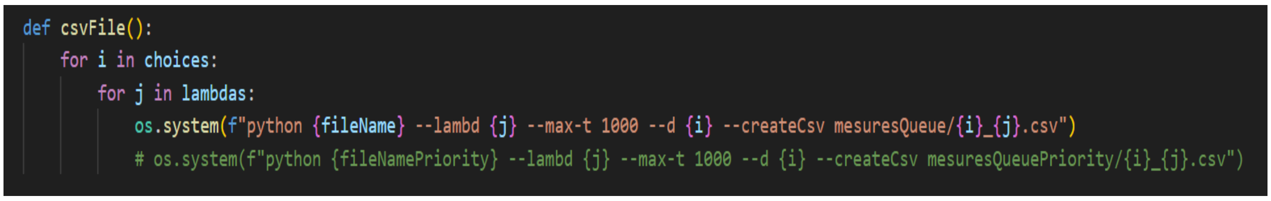


Figure 2: Results with n = 100 and measurements taken when the job ID is even.

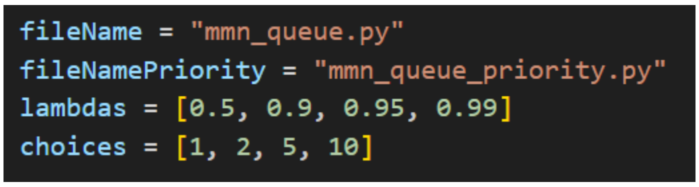
It is evident that as the size of the subset of d servers increases, the average queue length decreases. This is due to the ability to choose the shortest queue in which to place a job.

To generate these graphs, I created a dedicated file called "plotGraph.py," which was used for both this version of the assignment and the extension.

In the csvFile function, the scripts mmn_queue.py or mmn_queue_piority.py (for the extension) are executed, and CSV files are created for each λ and d.

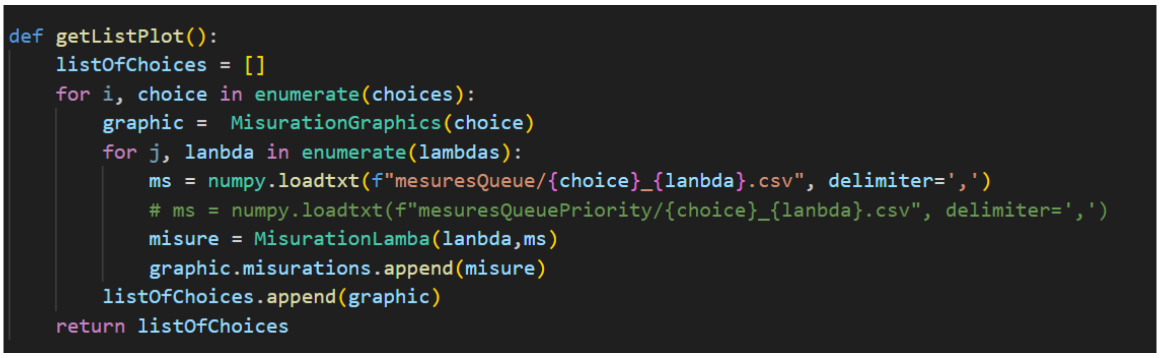


```
def csvFile():
    for i in choices:
        for j in lambdas:
            os.system(f"python {fileName} --lambd {j} --max-t 1000 --d {i} --createCsv mesuresQueue/{i}_{j}.csv")
            # os.system(f"python {fileNamePriority} --lambd {j} --max-t 1000 --d {i} --createCsv mesuresQueuePriority/{i}_{j}.csv")
```



```
fileName = "mmn_queue.py"
fileNamePriority = "mmn_queue_priority.py"
lambdas = [0.5, 0.9, 0.95, 0.99]
choices = [1, 2, 5, 10]
```

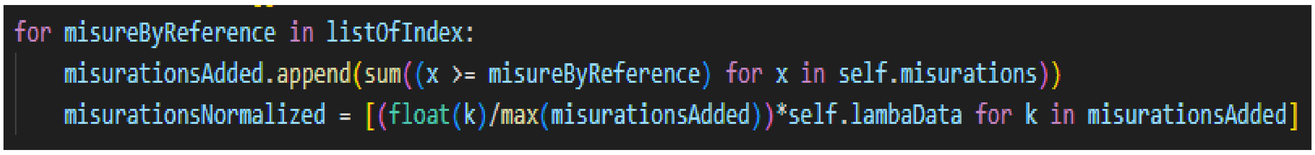
In the getListPlot function, for each element 'd' in the choices list, a MeasurementGraphics object was created, consisting of a 'd' and a list of measurements for that 'd'. Each measurement is a MeasurementLambda object, consisting of λ and the list of normalized measurements.



```python
def getListPlot():
    listOfChoices = []
    for i, choice in enumerate(choices):
        graphic =  MisurationGraphics(choice)
        for j, lanbda in enumerate(lambdas):
            ms = numpy.loadtxt(f"mesuresQueue/{choice}_{lanbda}.csv", delimiter=',')
            # ms = numpy.loadtxt(f"mesuresQueuePriority/{choice}_{lanbda}.csv", delimiter=',')
            misure = MisurationLamba(lanbda,ms)
            graphic.misurations.append(misure)
        listOfChoices.append(graphic)
    return listOfChoices
```

To normalize the measurements, I have:

- taken all possible indices from 1 to 15
- from the list of measurements, summed all elements greater than or equal to that index
- each sum was divided by the maximum of the sums and multiplied by λ



```python
for misureByReference in listOfIndex:
    misurationsAdded.append(sum((x >= misureByReference) for x in self.misurations))
    misurationsNormalized = [(float(k)/max(misurationsAdded))*self.lambaData for k in misurationsAdded]
```

In this way, the above-mentioned graphs were obtained.

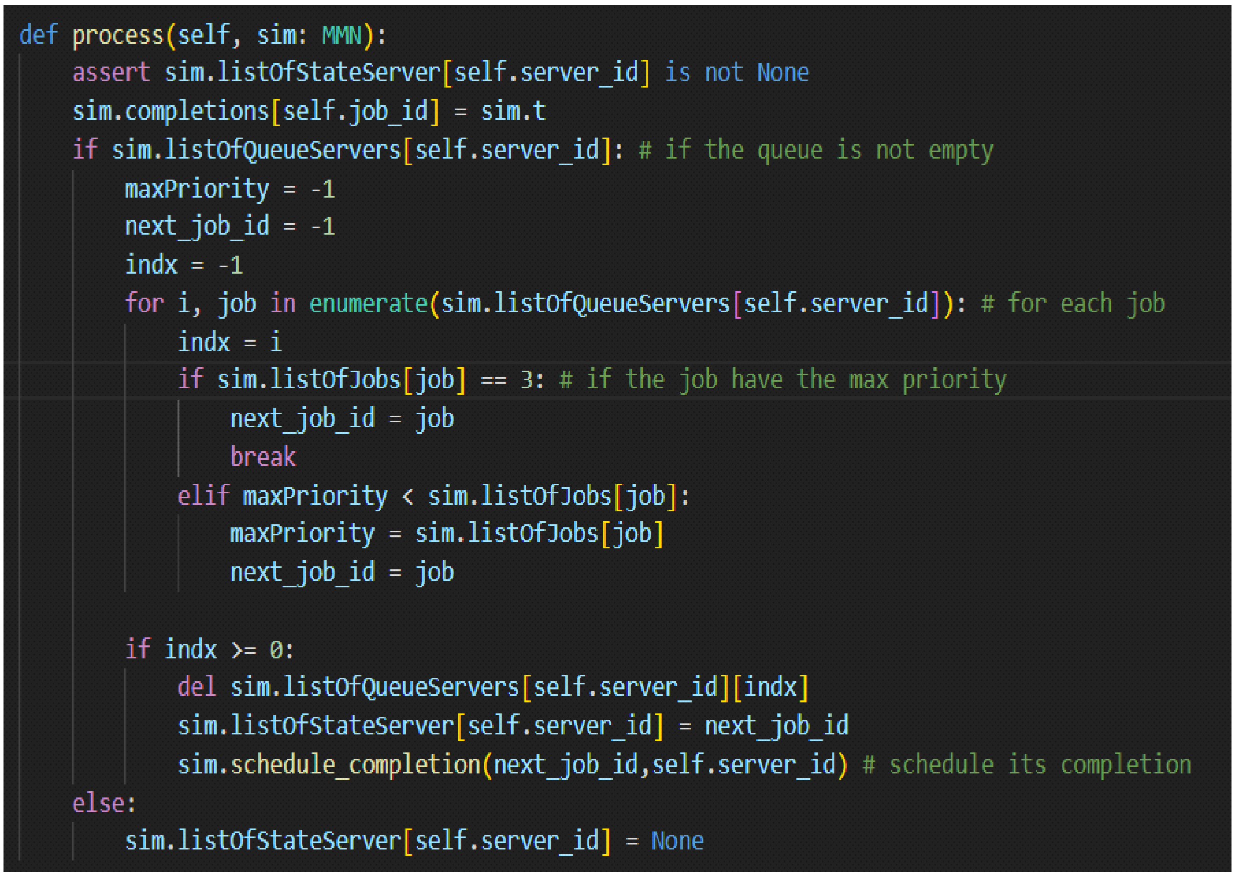
The same file was also used for the extension of the assignment by simply uncommenting the lines related to the file mmn_queue_priority.py in the csvFile and getListPlot functions and commenting out the others.

Simulator extension:

I chose to implement a priority queue for each server because in the real world, when jobs are created, they almost never have the same priority.

To accomplish this, I assigned priorities to jobs at the time of their creation, ranging from 0 to 3, where 3 is the highest priority.

The queues of the various servers were not reordered based on job priority, but rather, when a job needs to be selected to schedule its completion, the one with the highest priority is chosen.

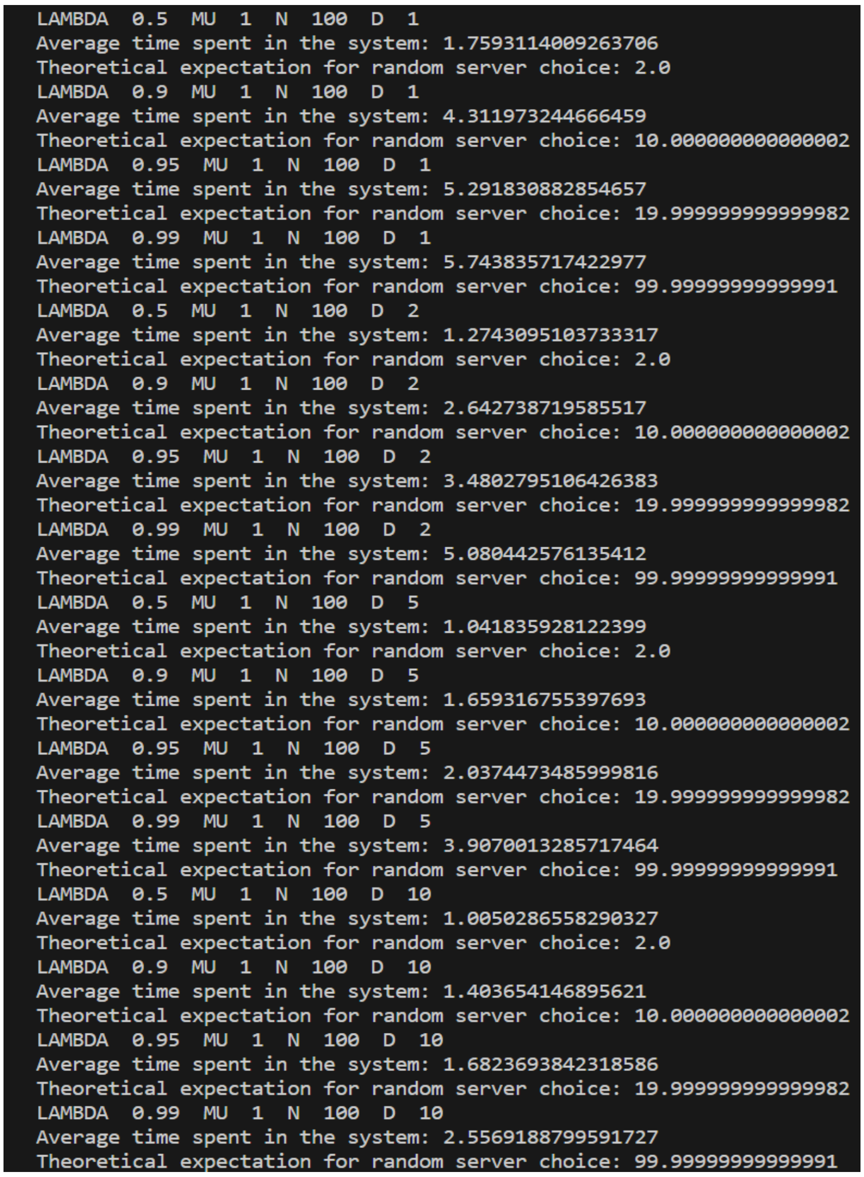


```python
def process(self, sim: MMN):
    assert sim.listOfStateServer[self.server_id] is not None
    sim.completions[self.job_id] = sim.t
    if sim.listOfQueueServers[self.server_id]: # if the queue is not empty
        maxPriority = -1
        next_job_id = -1
        indx = -1
        for i, job in enumerate(sim.listOfQueueServers[self.server_id]): # for each job
            indx = i
            if sim.listOfJobs[job] == 3: # if the job have the max priority
                next_job_id = job
                break
            elif maxPriority < sim.listOfJobs[job]:
                maxPriority = sim.listOfJobs[job]
                next_job_id = job

        if indx >= 0:
            del sim.listOfQueueServers[self.server_id][indx]
            sim.listOfStateServer[self.server_id] = next_job_id
            sim.schedule_completion(next_job_id,self.server_id) # schedule its completion
    else:
        sim.listOfStateServer[self.server_id] = None
```

In this way, the lengths of the server queues remained the same, but the average time spent in the system by various jobs decreased.

These are the results I obtained:



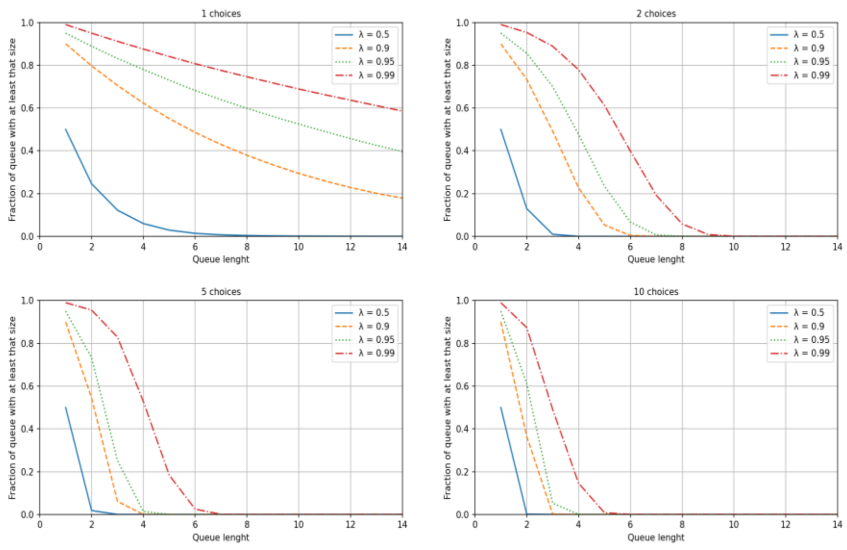
```
LAMBDA  0.5  MU  1  N  100  D  1
Average time spent in the system: 1.7593114009263706
Theoretical expectation for random server choice: 2.0
LAMBDA  0.9  MU  1  N  100  D  1
Average time spent in the system: 4.311973244666459
Theoretical expectation for random server choice: 10.000000000000002
LAMBDA  0.95  MU  1  N  100  D  1
Average time spent in the system: 5.291830882854657
Theoretical expectation for random server choice: 19.999999999999982
LAMBDA  0.99  MU  1  N  100  D  1
Average time spent in the system: 5.743835717422977
Theoretical expectation for random server choice: 99.99999999999991
LAMBDA  0.5  MU  1  N  100  D  2
Average time spent in the system: 1.2743095103733317
Theoretical expectation for random server choice: 2.0
LAMBDA  0.9  MU  1  N  100  D  2
Average time spent in the system: 2.642738719585517
Theoretical expectation for random server choice: 10.000000000000002
LAMBDA  0.95  MU  1  N  100  D  2
Average time spent in the system: 3.4802795106426383
Theoretical expectation for random server choice: 19.999999999999982
LAMBDA  0.99  MU  1  N  100  D  2
Average time spent in the system: 5.080442576135412
Theoretical expectation for random server choice: 99.99999999999991
LAMBDA  0.5  MU  1  N  100  D  5
Average time spent in the system: 1.041835928122399
Theoretical expectation for random server choice: 2.0
LAMBDA  0.9  MU  1  N  100  D  5
Average time spent in the system: 1.659316755397693
Theoretical expectation for random server choice: 10.000000000000002
LAMBDA  0.95  MU  1  N  100  D  5
Average time spent in the system: 2.0374473485999816
Theoretical expectation for random server choice: 19.999999999999982
LAMBDA  0.99  MU  1  N  100  D  5
Average time spent in the system: 3.9070013285717464
Theoretical expectation for random server choice: 99.99999999999991
LAMBDA  0.5  MU  1  N  100  D  10
Average time spent in the system: 1.0050286558290327
Theoretical expectation for random server choice: 2.0
LAMBDA  0.9  MU  1  N  100  D  10
Average time spent in the system: 1.403654146895621
Theoretical expectation for random server choice: 10.000000000000002
LAMBDA  0.95  MU  1  N  100  D  10
Average time spent in the system: 1.6823693842318586
Theoretical expectation for random server choice: 19.999999999999982
LAMBDA  0.99  MU  1  N  100  D  10
Average time spent in the system: 2.5569188799591727
Theoretical expectation for random server choice: 99.99999999999991
```

It can be observed that in the case of d=1, the average time of jobs spent in the priority queue significantly decreases, for example, in the case of λ=0.99, from 20.966603482220066 in the non-priority queue to 5.743835717422977 in the priority one.

The other measurements remain similar to those taken with non-priority queues because the queue lengths are much shorter.

As for the assignment without the priority queue, I then generated the graphs to observe the average length of the queues.

The graphs obtained were generated with n=100, taking measurements every time jobs with even IDs entered the server queues.



The priority queue is certainly very useful in cases where I need to schedule many jobs in a single queue, to expedite the more important ones.