Exercise 4: Simulation of M/M/1 Queues

Arnav Mahajan
EP21B004

I. Introduction

This exercise aims to simulate M/M/1 queues with different service schemes such as - Vanilla (FIFO), LIFO, and Processor Sharing (PS). For all of the schemes, a total of 10,000 time slots have been considered with a Poisson Arrival process and Exponential Services.

The language used for the exercise is **Python**.

II. Implementation

Common Parts of Implementation: A queue using the Python list data type has been implemented. The values of lists are used to store the time spent by the element in the queue. The Poisson and Exponential random variables are generated using `numpy.random` libraries with required mean arrival and service times.

Outer for loop iterates over all the values of λ, inside which another for loop iterates over each time slot (10,000 slots total). At each time slot, poison and exponential random variables are created. All the elements inside the queue are incremented by one. A required number of arrival elements, as dictated by the Poisson random variable, are added to the head of the queue using the `queue.insert(0, 0)` command.

The implementation of service schemes is discussed below.

A. Vanilla Service Scheme

After adding the required number of elements at the head, the number of elements, dictated by an exponential random variable, is removed from the tail of the queue using `queue.pop()` function. Since each element stores the 'wait time', it is appended into necessary lists for later statistics. If no element is present/the number of elements is less than the number of services, operation continues as normal.

B. LIFO Service Scheme

All commands are the same as that for *Vanilla Service Scheme*, except that every new arrival is appended at the tail of the queue using `queue.append(0)` command.

C. Processor Sharing Scheme

In this case, another list (queue) is maintained, which stores the remaining work for each of the queue elements. Each element starts with a unit work amount. Thus, for every new arrival, `work.insert(0, 1)` command is executed.

A number of services, dictated by an exponential distribution, is given equally to all the elements present. This is done as: `work[i] -= num_services[t]/n`. If the work remaining for any element is ≤ 0, then that element and the corresponding work element are popped off from the list. Loop indices are adjusted accordingly.

To calculate the mean `statistics.mean()` is used. All the plots are obtained using Matplotlib.

III. Simulation Results

A. Vanilla Service Scheme

Sr. No.	Offered Load (λ/μ)	Mean Queue Length	Mean Wait Time
1	0.1	0.170	0.169
2	0.2	0.372	0.248
3	0.3	0.624	0.311
4	0.4	1.046	0.416
5	0.5	1.577	0.524
6	0.6	2.457	0.700
7	0.7	3.859	0.964
8	0.8	7.318	1.625
9	0.9	15.453	3.090

TABLE I: Simulation Data for Vanilla Service Scheme

B. LIFO Service Scheme

Sr. No.	Offered Load (λ/μ)	Mean Queue Length	Mean Wait Time
1	0.1	0.179	0.179
2	0.2	0.348	0.232
3	0.3	0.593	0.298
4	0.4	0.991	0.396
5	0.5	1.553	0.518
6	0.6	2.425	0.694
7	0.7	3.970	0.992
8	0.8	7.231	1.604
9	0.9	15.625	3.117

TABLE II: Simulation Data for LIFO Service Scheme

C. Processor Sharing Service Scheme

Sr. No.	Offered Load (λ/μ)	Mean Queue Length	Mean Wait Time
1	0.1	0.275	0.280
2	0.2	0.607	0.407
3	0.3	1.186	0.595
4	0.4	2.097	0.840
5	0.5	3.586	1.198
6	0.6	5.951	1.700
7	0.7	10.448	2.616
8	0.8	22.020	4.895
9	0.9	59.889	11.980

TABLE III: Simulation Data for Processor Sharing Service Scheme

IV. Plots

A. Vanilla Service Scheme

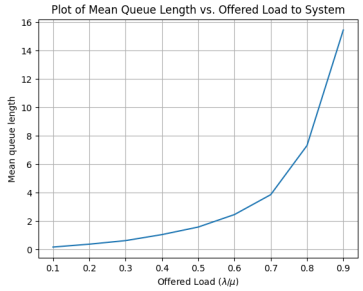

Fig. 1: Mean Queue Length - Vanilla

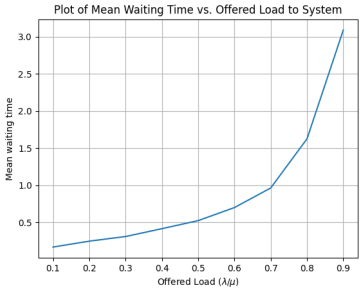

Fig. 2: Mean Wait Time - Vanilla

B. LIFO Service Scheme

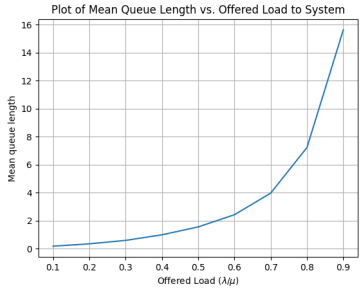

Fig. 3: Mean Queue Length - LIFO

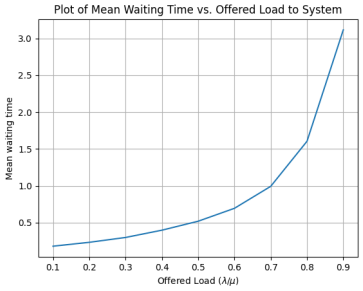

Fig. 4: Mean Wait Time - LIFO

C. Processor Sharing Service Scheme

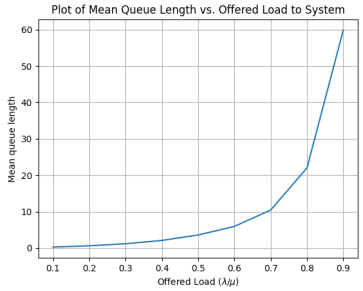

Fig. 5: Mean Queue Length - Processor Sharing

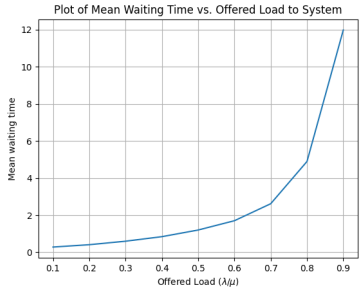

Fig. 6: Mean Wait Time - Processor Sharing

V. Analysis

It is seen that the Vanilla and LIFO schemes follow very similar curves. This is because even though the head (end) elements in a LIFO queue spend much more time than Vanilla, the mean wait time is compensated by the tail elements. Queue length, as expected, is not affected from Vanilla to LIFO.

It is also seen that both the waiting time statistics and the queue lengths for processor-sharing schemes are much higher than the other two. This can be attributed to the fact that all the work is distributed equally among all the elements, even if any of the elements do not require the given amount.