CENX 570: Simulation and Modelling

Dr Nasser Eddine Rikli

December 2023

Project: Phase 2

M/M/1 queues and variations

Submitted By:

Mohammed Shahzad

444105788@student.ksu.edu.sa

Introduction to M/M/1 queues

In queueing theory, a discipline within the mathematical theory of probability, an M/M/1 queue represents the queue length in a system having a single server, where arrivals are determined by a Poisson process and job service times have an exponential distribution. The model's name is written in Kendall's notation. The model is the most elementary of queueing models and an attractive object of study as closed-form expressions can be obtained for many metrics of interest in this model. An extension of this model with more than one server is the M/M/c queue.

Components of Queueing System

If any customer is willing to get a service, s/he should check whether a server is idle or not. If the server is vacant, customer gets the service immediately. However, if at least one customer is waiting for the service in front of each of the servers, then the new arrival should line up. Figure 1 represents the basic queueing model where the procedure of a simple queueing system is shown.

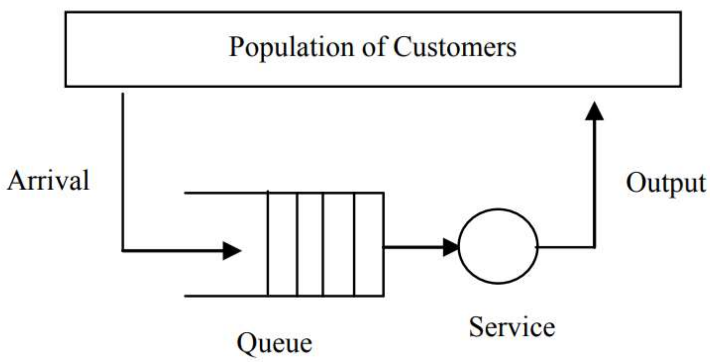
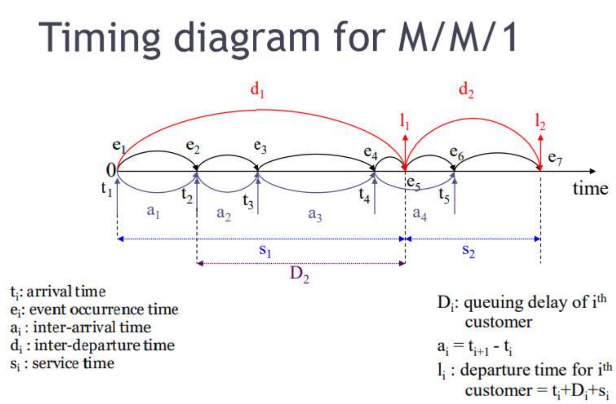
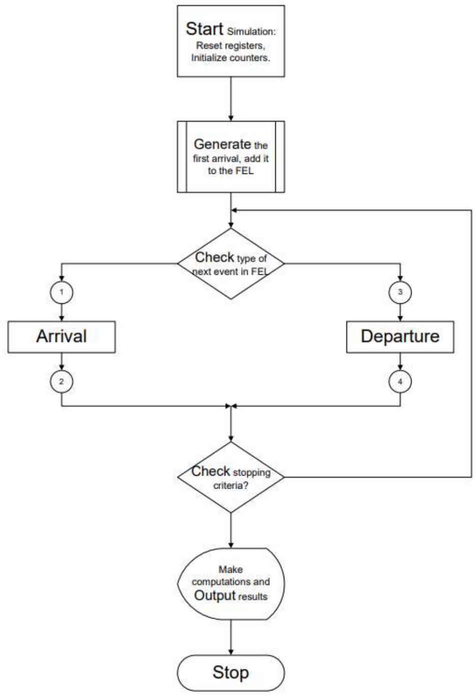


Figure 1. General queueing system

From the time someone starts standing in a queue until getting served, there are certain steps to follow. These steps are called the components of a queue which are characterized by the arrival process of customers, behaviour of customers, service times, service discipline and service capacity [1].



Flowchart for a simple M/M/1 queue



Basic Flow-Chart of an M/M/1 Queueing System

Important notation for M/M/1 queuing formulae

• λ : mean rate of arrival and equals 1/E[Inter-arrival-Time]
• μ : mean service rate and equals 1/E[Service-Time]
• $\rho = \lambda/(c\mu)$: utilization of the server
• L : mean number of customers in the system
• L_q : mean number of customers in the queue
• W : mean waiting time in the system
• W_q : mean waiting time in the queue

M/M/1 Queues examples

TASK 1. Using the simulation program of the M/M/1 queue as a guide:

(a) Test the simulation model by evaluating the average delay and the average queue size for the following sets of parameters: $m\lambda = \mu$ with $\lambda = 1, 2$ packets/sec and $m = 1, 2, 3, 4$.

(b) Comment on your results.

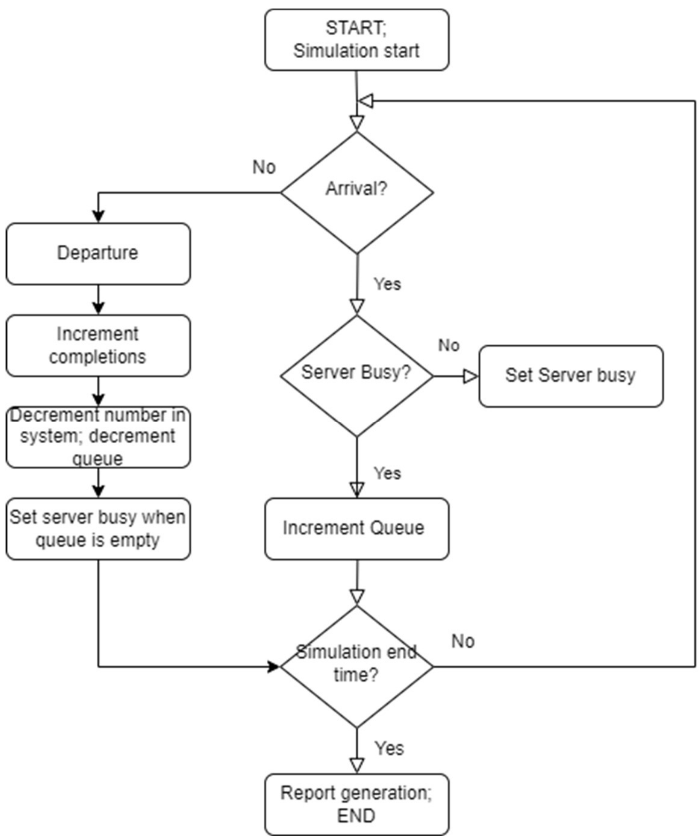
Answer:

Given: When $\lambda = 1,2$; $\mu = m\lambda$; $m = [1,2.3,4]$, total 8 cases

Algorithm (pseudocode)

```
1.  Input: Arrival time, service time, Queue max
2.  Output: Server utilization, mean no of customers in queue, mean time in
    queue, etc
3.  Start simulation clock
4.  If arrival event? → Yes: Next step; No: Step 7
5.  Check if number of customers in system is more than 1
6.  Server is busy? Yes: Increment queue; No: Set server busy
7.  Departure event
8.  Schedule departure
9.  Increment no of completions; Decrease queue
10. Simulation end time? No: loop back to Step 4, Yes: Step 11
11. Generate report
12. END
```

Flowchart



Flowchart for M/M/1 queuing system

Analytical Solution [2]

See "Important notations for M/M/1 queueing formulae" above

<u>Server Utilization (ρ)</u>

$$\rho = \lambda/\mu$$

<u>Mean number of customers in system (L)</u>

$L = \rho/(1-\rho)$ | $L = \lambda W$

Mean time delay in the system (W)

$W = \rho(1/\mu)/(1-\rho)$ | $W = Wq + 1/\mu$

Mean number in the queue (L_q)

$L_q = \rho^2/(1-\rho)$

Mean time in queue (W_q)

$W_q = L_q/\lambda$

For: m = 1,2,3,4

$\lambda = 1$; $\mu = 1$; ➔ $\rho = 1.00$ or 100%; L = infinite; L_q = infinite; W = infinite ; W_q = infinite

$\lambda = 1$; $\mu = 2$; ➔ $\rho = 0.50$ or 50%; L = 1; $L_q = 0.5$; W = 1 ; $W_q = 0.5$

$\lambda = 1$; $\mu = 3$; ➔ $\rho = 0.33$ or 33%; L = 0.5; $L_q = 0.1667$; W = 0.5 ; $W_q = 0.1667$

$\lambda = 1$; $\mu = 4$; ➔ $\rho = 0.25$ or 25%; L = 0.3333; $L_q = 0.0833$; W = 0.3333 ; $W_q = 0.0833$

$\lambda = 2$; $\mu = 2$; ➔ $\rho = 1$ or 100%; L = infinite; L_q = infinite; W = infinite ; W_q = infinite

$\lambda = 2$; $\mu = 4$; ➔ $\rho = 0.5$ or 50%; L = 1; $L_q = 0.5$; W = 0.5 ; $W_q = 0.25$

$\lambda = 2$; $\mu = 6$; ➔ $\rho = 0.3333$ or 33.33%; L = 0.5; $L_q = 0.1667$; W = 0.25 ; $W_q = 0.0833$

$\lambda = 2$; $\mu = 8$; ➔ $\rho = 0.25$ or 25%; L = 0.3333; $L_q = 0.0833$; W = 0.1667 ; $W_q = 0.0417$

Results Obtained

When arrival rate (λ) = 1, m= 1,2,3,4; Total simulated time = 1000000000 sec (1.0e9)

For Mean service rate (μ) = m λ	1	2	3	4
Server utilization ρ %	92.418677 %	49.998932 %	33.334189 %	25.000554 %
Mean number of customers in system (L)	25801.08	0.999394	0.499848	0.333259
Mean number in the queue (L_q)	25800.08	0.499405	0.166506	0.083253
Mean time delay in the system (W)	25798.00	0.999251	0.499765	0.333204
Mean time in queue (Wq)	12898.67	0.333028	0.124908	0.066617

When arrival rate (λ) = 2, m= 1,2,3,4; Total simulated time = 1000000000 sec (1.0e9)

For Mean service rate (μ) = m λ	2	4	6	8
Server utilization ρ %	96.092036 %	49.999560 %	33.334509 %	25.000677 %
Mean number of customers in system (L)	21292.702	0.999416	0.499852	0.333260

Mean number in the queue (L_q)	21291.701	0.499420	0.166507	0.083253
Mean time delay in the system (W)	10644.9672	0.499634	0.249884	0.166602
Mean time in queue (Wq)	5322.3248	0.166518	0.062454	0.033308

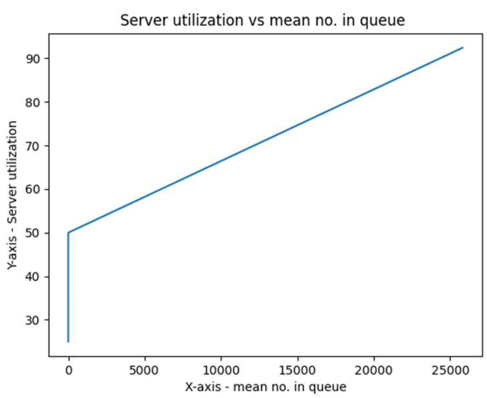
Results Discussion

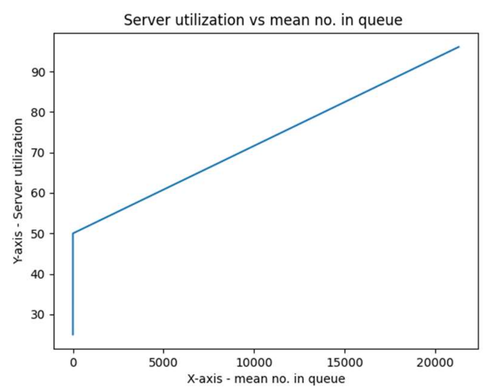
We run the simulation for 1.0e9, so to remove transient conditions. As we observe, with the service rate increase, the server utilization decreases. This is because the server is idle when waiting for arrival of customers. The mean time in queue and mean customers in system also decrease with increase in service rate. When the server is idle, packets are serviced with less waiting time, therefore the mean time in queue also decreases.

Our simulation results align with the expected analytical results which is a positive sign that our simulation shows positive correlation with expected real-world systems.

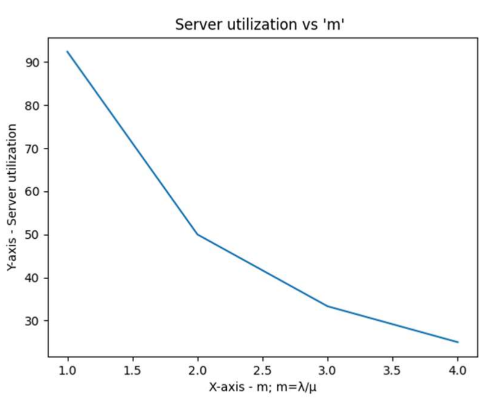
Graphs

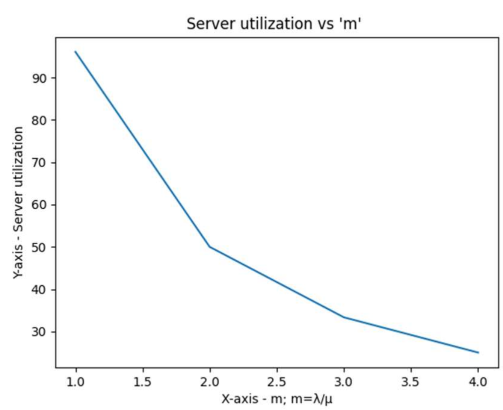
Server utilization on y-axis vs queue size on x-axis





Server utilization on y-axis vs 'm' on x-axis





Variations in M/M/1 Queue

TASK 2. Consider the following two variations of the previous model:

- Upon the arrival of a customer for service, if the server is found busy, it checks the current queue size, and if it is greater or queue to a size Qmax it is dropped, otherwise it joins the queue.
- Instead of having one server, assume that you have two independent and identical servers operating in parallel; so an arriving customer will join the queue only if both queues are busy, and if a customer arrives and finds both servers idle, he is served randomly by one server with equal probability.

(a) Run the simulation model for the two previous systems. Gather the necessary statistics such as: average queue size, average queuing delay. Try to experiment with $m\lambda = \mu$ ($m = 1, 2, 3, 4$), and various parameter values of $Q_{max} = 10, 20, 30, 40, 50$. Consider the following units for Q_{max}:

- packets;
- kbytes with a transmission rate R = 10 Mbps.

(b) Compare your results with those in (1), and give any comments. (Use tables and graphs to support your assertions).

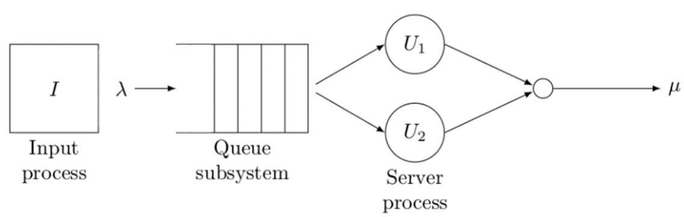
3. In all previous cases:

(a) use the average queue and system sizes, the average queueing and total delays, and server utilization as performance criteria.

(b) give the expected analytical results (optional)

Answer:

➔ **M/M/K with 2 Servers**



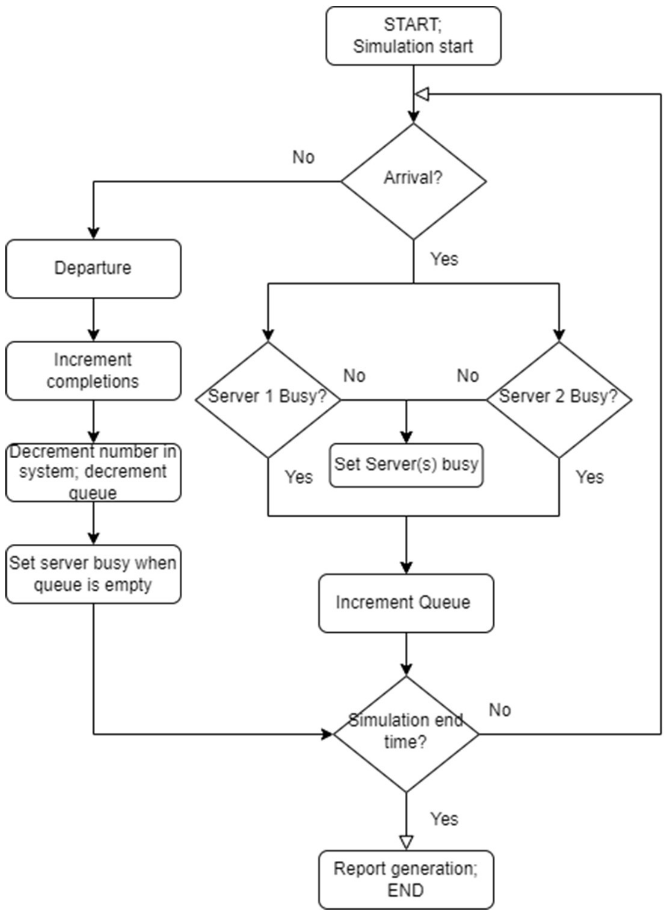
M/M/2 queuing system

Algorithm (Pseudocode)

```
1. Input: Arrival time, service time, Queue max
2. Output: Server utilization, mean no of customers in queue, mean time in
   queue, etc
3. Start simulation clock
4. If arrival event? → Yes: Next step; No: Step 9
5. Check if number of customers in system is more than 1
6. If 2 Server is busy: → Yes: Next step; No: Set server(s) busy
7. Increment queue
```

```
8.  Departure event
9.  Schedule departure
10. Increment no of completions
11. Simulation end time? No: loop back to Step 4, Yes: Step 13
12. Generate report
13. END
```

Flowchart



M/M/2 queuing system

Analytical Solution [2][3]

Analytical Solutions for M/M/2

<u>Server Utilization (ρ)</u>

$\rho = \lambda/c\mu$ where c is number of servers

<u>Mean number of customers in system (L) [3]</u>

$L = \lambda^3 / \mu(4\mu^2 - \lambda^2)$ [2]

<u>Mean time delay in the system (W) [3]</u>

$W = \lambda^2 / \mu(4\mu^2 - \lambda^2)$

For: m = 1,2,3,4 ; servers = 2;

$\lambda = 1$; $\mu = 1$; ➔ $\rho = 0.5$ or 50%; $L = 1.3333$; $L_q = 0.3333$; $W = 1.3333$; $W_q = 0.3333$

$\lambda = 1$; $\mu = 2$; ➔ $\rho = 0.25$ or 25%; $L = 0.5333$; $L_q = 0.0333$; $W = 0.5333$; $W_q = 0.0333$

$\lambda = 1$; $\mu = 3$; ➔ $\rho = 0.1667$ or 16.67%; $L = 0.3429$; $L_q = 0.0095$; $W = 0.3429$; $W_q = 0.0095$

$\lambda = 1$; $\mu = 4$; ➔ $\rho = 0.125$ or 12.5%; $L = 0.254$; $L_q = 0.004$; $W = 0.254$; $W_q = 0.004$

$\lambda = 2$; $\mu = 2$; ➔ $\rho = 0.5$ or 50%; $L = 1.3333$; $L_q = 0.3333$; $W = 0.6667$; $W_q = 0.1667$

$\lambda = 2$; $\mu = 4$; ➔ $\rho = 0.25$ or 25%; $L = 0.5333$; $L_q = 0.0333$; $W = 0.2667$; $W_q = 0.0167$

$\lambda = 2$; $\mu = 6$; ➔ $\rho = 0.1667$ or 16.67%; $L = 0.3429$; $L_q = 0.0095$; $W = 0.1714$; $W_q = 0.0048$

$\lambda = 2$; $\mu = 8$; ➔ $\rho = 0.125$ or 12.5%; $L = 0.254$; $L_q = 0.004$; $W = 0.127$; $W_q = 0.002$

Results Obtained

When arrival rate (λ) = 1, m= 1,2,3,4; Total simulated time = 1000000000 sec (1.0e9), 2 Servers

For Mean service rate (μ) = m λ	1	2	3	4
Server utilization ρ %	69.199079 %	23.124429 %	12.175665 %	7.627528 % %
Mean number of customers in system (L)	1.331069	0.542706	0.348671	0.257693
Mean number in the queue (L_q)	0.165445	0.042954	0.015695	0.007857
Mean time delay in the system (W)	1.328934	0.542017	0.348328	0.257386
Mean time in queue (Wq)	0.165179	0.042900	0.015679	0.007848

When arrival rate (λ) = 2, m= 1,2,3,4; Total simulated time = 1000000000 sec (1.0e9), 2Servers

For Mean service rate (μ) = m λ	2	4	6	8
Server utilization ρ %	69.269680 %	23.189368 %	12.190971 %	7.635562 %
Mean number of customers in system (L)	1.333644	0.543800	0.349319	0.257953
Mean number in the queue (L_q)	0.332547	0.043351	0.015808	0.007817
Mean time delay in the system (W)	0.665841	0.271525	0.174451	0.128822
Mean time in queue (Wq)	0.166029	0.021646	0.007895	0.003904

Results Discussion

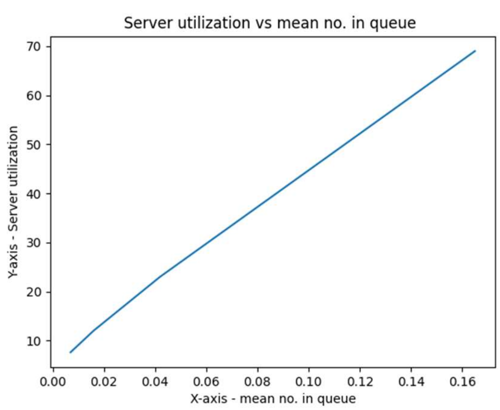
We run the simulation for 1.0e9, so to remove transient conditions. As we observe, with the service rate increase, the server utilization decreases. This is because the server is idle when waiting for arrival of customers. The mean time in queue and mean customers in system also decrease with increase in service rate. When the server is idle, packets are serviced with less waiting time, therefore the mean time in queue also decreases.

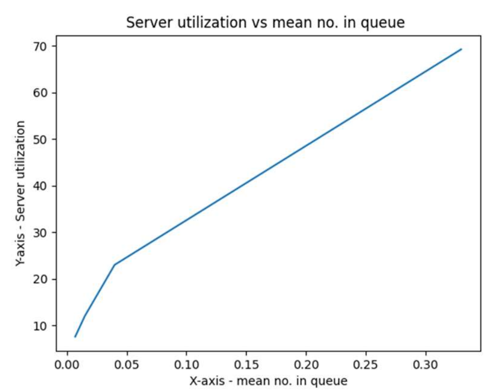
Our simulation results align with the expected analytical results which is a positive sign that our simulation shows positive correlation with expected real-world systems.

Clearly, we see that due to 2 servers, we have better throughput and lesser mean time spent in queue, mean time in system is reduced when compared to simple m/m/1 queueing system.

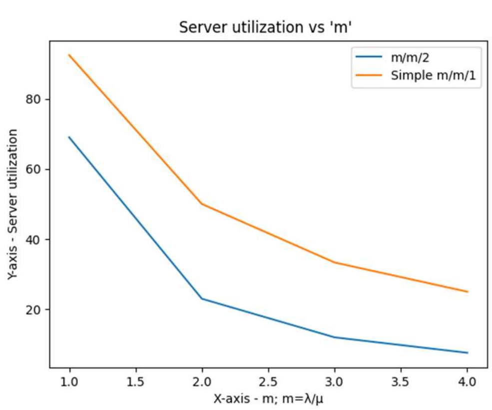
Graphs

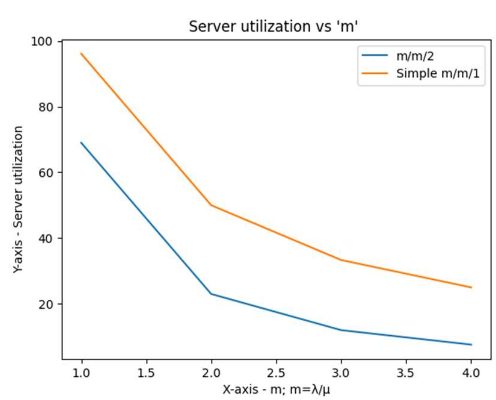
Server utilization on y-axis vs queue size on x-axis





Server utilization on y-axis vs ‘m’ on x-axis and simple MM1



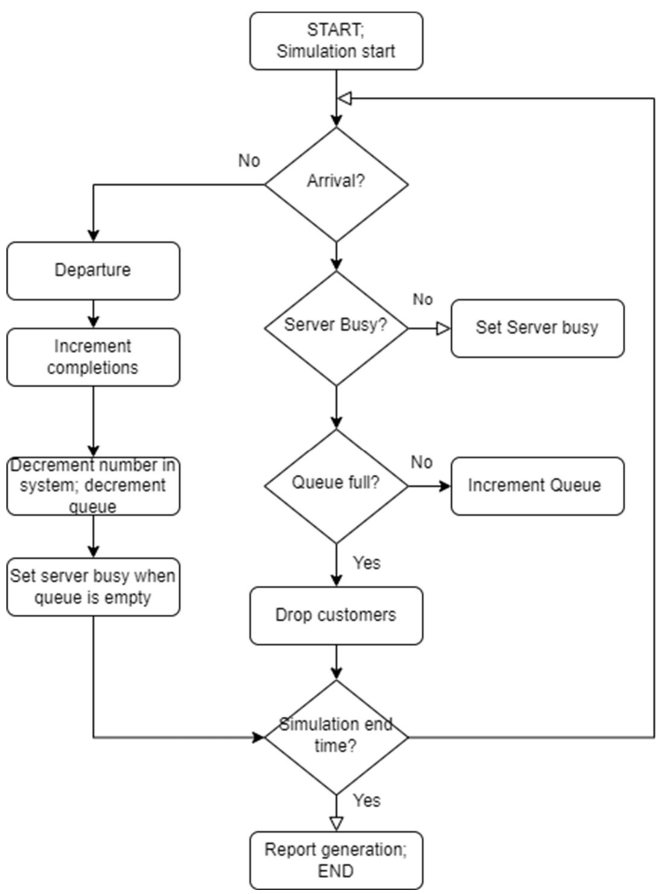


➔ **M/M/1 with Queue Size in no. of arrivals**

Algorithm (Pseudocode)

```
1.  Input: Arrival time, service time, Queue max
2.  Output: Server utilization, mean no of customers in queue, mean time in
    queue, etc
3.  Start simulation clock
4.  If arrival event? → Yes: Next step; No: Step 9
5.  Check if number of packets in system is more than 1
6.  If Server is busy: → Yes: Next step; No: Set server busy
7.  If queue full: Yes: Drop packets to fit; No: Next step
8.  Increment queue
9.  Departure event
10. Schedule departure
11. Increment no of completions
12. Simulation end time? No: loop back to Step 4, Yes: Step 13
13. Generate report
14. END
```

Flowchart



M/M/2 queuing system

Results Obtained

Qmax [10,20,30,40,50], (μ) = m λ, m =[1,2,3,4] = Total 40 cases

When arrival rate (λ) = 1, m= [1:4]; Total simulated time = 1000000 sec (1.0e6), and Q_{max}

Qmax = 10	For Mean service rate (μ) = m λ	1	2	3	4
	Server utilization ρ %	92.230007 %	50.143037 %	33.306606 %	24.952303 %
	Mean number of customers in system (L)	5.995071	1.006785	0.499632	0.332313
	Mean number in the queue (L_q)	4.643975	0.504808	0.166539	0.082789
	Mean time delay in the system (W)	6.491467	1.003841	0.499068	0.332125
	Mean time in queue (Wq)	2.742365	0.334777	0.124669	0.066278
	Customers dropped	155650	266	8	0

Qmax = 20	For Mean service rate (μ) = m λ	1	2	3	4
	Server utilization ρ %	95.618653 %	50.094592 %	33.306926 %	24.952303 %
	Mean number of customers in system (L)	10.971433	1.004316	0.499666	0.332313
	Mean number in the queue (L_q)	9.561517	0.503370	0.166597	0.082789
	Mean time delay in the system (W)	11.451480	1.001815	0.499100	0.332125
	Mean time in queue (Wq)	5.235462	0.333854	0.124689	0.066278
	Customers dropped	86790	0	0	0

	For Mean service rate (μ) = m λ	1	2	3	4

Qmax = 30	Server utilization ρ %	96.874007 %	50.094592 %	33.306926 %	24.952303 %
	Mean number of customers in system (L)	15.925432	1.004316	0.499666	0.332313
	Mean number in the queue (L_q)	14.487437	0.503370	0.166597	0.082789
	Mean time delay in the system (W)	16.407084	1.001815	0.499100	0.332125
	Mean time in queue (Wq)	7.716019	0.333854	0.124689	0.066278
	Customers dropped	60822	0	0	0

Qmax = 40	For Mean service rate (μ) = m λ	1	2	3	4
	Server utilization ρ %	97.536393 %	50.094592 %	33.306926 %	24.952303 %
	Mean number of customers in system (L)	20.888867	1.004316	0.499666	0.332313
	Mean number in the queue (L_q)	19.434777	0.503370	0.166597	0.082789
	Mean time delay in the system (W)	21.370152	1.001815	0.499100	0.332125
	Mean time in queue (Wq)	10.186488	0.333854	0.124689	0.066278
	Customers dropped	46928	0	0	0

Qmax = 50	For Mean service rate (μ) = m λ	1	2	3	4
	Server utilization ρ %	97.834987 %	50.094592 %	33.306926 %	24.952303 %
	Mean number of customers in system (L)	25.849994	1.004316	0.499666	0.332313
	Mean number in the queue (L_q)	24.409649	0.503370	0.166597	0.082789

	Mean time delay in the system (W)	26.299076	1.001815	0.499100	0.332125
	Mean time in queue (Wq)	12.674155	0.333854	0.124689	0.066278
	Customers dropped	35942	0	0	0

When arrival rate (λ) = 2, m= [1:4]; Total simulated time = 1000000 sec (1.0e6), Q_{max} = 50

Qmax =10	For Mean service rate (μ) = m λ	2	4	6	8
	Server utilization ρ %	92.189958 %	50.159324 %	33.347293 %	24.994069 %
	Mean number of customers in system (L)	5.977945	1.007889	0.500483	0.333059
	Mean number in the queue (L_q)	4.633653	0.505771	0.166997	0.083114
	Mean time delay in the system (W)	3.236545	0.502733	0.249958	0.166393
	Mean time in queue (Wq)	1.368712	0.167944	0.062429	0.033255
	Customers dropped	306528	524	8	4

Qmax =20	For Mean service rate (μ) = m λ	2	4	6	8
	Server utilization ρ %	95.621321 %	50.112467	33.347320 %	24.994070 %
	Mean number of customers in system (L)	10.949379	1.004541	0.500497	0.333062
	Mean number in the queue (L_q)	9.543972	0.503416	0.167024	0.083121
	Mean time delay in the system (W)	5.717339	0.501255	0.249964	0.166394
	Mean time in queue (Wq)	2.612667	0.167187	0.062435	0.033256
	Customers dropped	171544	0	0	0

Qmax =30	For Mean service rate (μ) = m λ	2	4	6	8
	Server utilization ρ %	96.849434 %	50.112467	33.347320 %	24.994070 %
	Mean number of customers in system (L)	15.906042	1.004541	0.500497	0.333062
	Mean number in the queue (L_q)	14.471261	0.503416	0.167024	0.083121
	Mean time delay in the system (W)	8.197517	0.501255	0.249964	0.166394
	Mean time in queue (Wq)	3.853325	0.167187	0.062435	0.033256
	Customers dropped	120762	0	0	0

Qmax =40	For Mean service rate (μ) = m λ	2	4	6	8
	Server utilization ρ %	97.524083 %	50.112467	33.347320 %	24.994070 %
	Mean number of customers in system (L)	20.812520	1.004541	0.500497	0.333062
	Mean number in the queue (L_q)	19.368237	0.503416	0.167024	0.083121
	Mean time delay in the system (W)	10.648870	0.501255	0.249964	0.166394
	Mean time in queue (Wq)	5.075753	0.167187	0.062435	0.033256
	Customers dropped	91452	0	0	0

	For Mean service rate (μ) = m λ	2	4	6	8
	Server utilization ρ %	98.129008 %	50.112467	33.347320 %	24.994070 %
	Mean number of customers in system (L)	25.940533	1.004541	0.500497	0.333062

Qmax =50	Mean number in the queue (L_q)	24.487903	0.503416	0.167024	0.083121
	Mean time delay in the system (W)	13.201199	0.501255	0.249964	0.166394
	Mean time in queue (Wq)	6.356665	0.167187	0.062435	0.033256
	Customers dropped	73460	0	0	0

Results Discussion

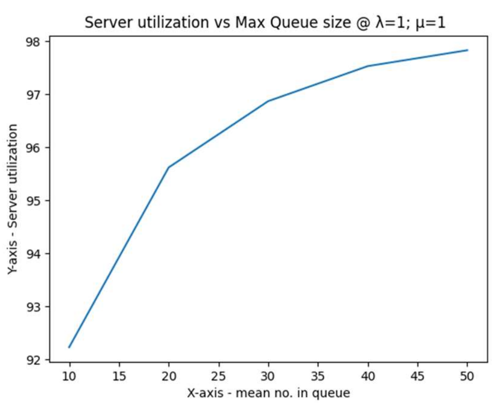
We run the simulation for 1.0e6, so to remove transient conditions. As service rate increases the server utilization decreases, because the server is idle when waiting for arrival. As queue size increase, eventually less packets are dropped. Packets dropped increases with increase in arrival rate, decrease with service rate. Server utilization increases with increase in queue size.

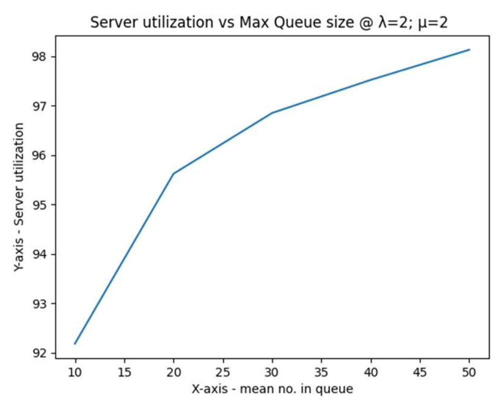
Our simulation results align with the expected analytical results which is a positive sign that our simulation shows positive correlation with expected real-world systems.

When compared to simple M/M/1, the queue size with constant or variable packets size performs better with increasing queue sizes. Even with restricted queue size, the performance and server utilization are slightly better than simple M/M/1 queueing system.

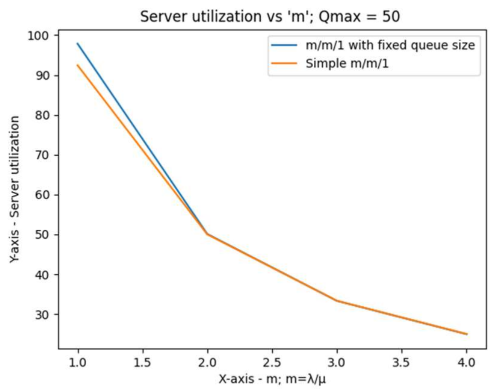
Graphs

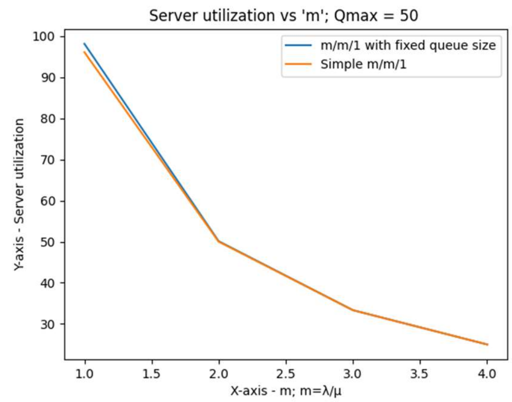
Server utilization on y-axis vs queue size on x-axis



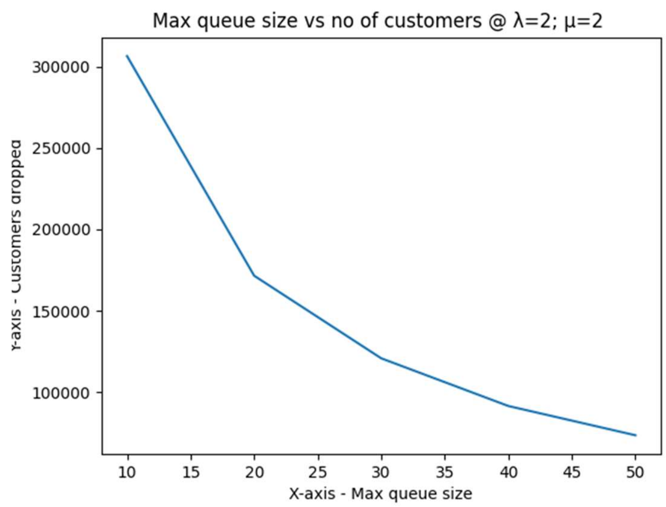


Server utilization on y-axis vs ‘m’ on x-axis and simple MM1





Queue size vs drops



- ➔ **M/M/1 with Queue Size in Kbytes**

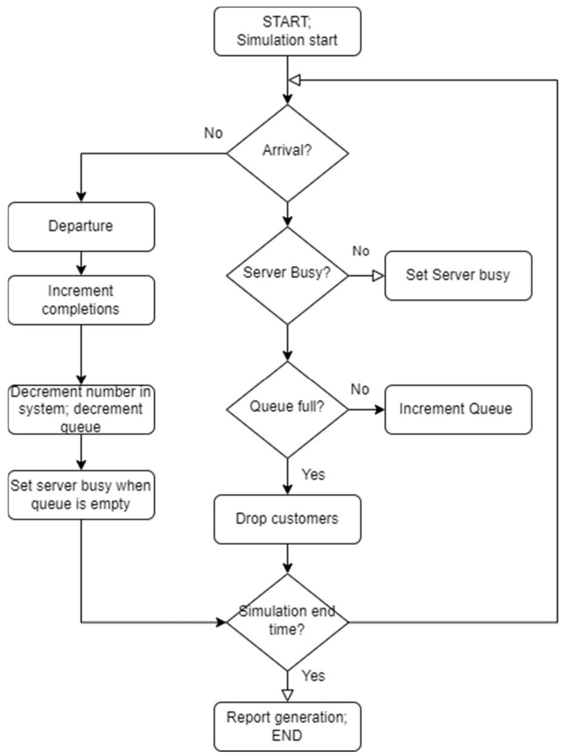
Note: Transmission rate R is 10Kbps

Algorithm (Pseudocode)

```
1. Input: Arrival time, service time, Queue max
2. Output: Server utilization, mean no of packets in queue, mean time in
   queue, etc
3. Start simulation clock
4. If arrival event? → Yes: Next step; No: Step 9
5. Check if number of packets in system is more than 1
6. If Server is busy: → Yes: Next step; No: Set server busy
7. If queue(in size Kb) full: Yes: Drop customers to fit; No: Next step
8. Increment queue
9. Departure event
10. Schedule departure
11. Increment no of completions
12. Simulation end time? No: loop back to Step 4, Yes: Step 13
13. Generate report
```

14. END

Flowchart



M/M/2 queuing system

Analytical Solution

Given data:

Transmission rate R = 10kbps; Queue size [10,20,30,40,50] Kb

- ⇨ Transmission rate R = 10 /8 Kbytes per sec = 1.25 Kbytes per sec
- ⇨ Lets say queue size [10,20,30,40,50] = 10 kb, and service rate = 1 packet/sec
- ⇨ Size of 1 packet = service time * transmission rate R = 1 sec/pac * 1.25 kbytes /sec
- ⇨ Size of 1 packet for given service time and transmission rate = 1.25 Kbytes/packet

When queue size is 10 Kb, and packet size is constant

- ⇨ Maximum queue size = queue size in kb / size of 1 packet
- ⇨ Max queue size = 10/1.25 = 8 packets

To validate, our analytical results we ran a M/M/1 simulation with fixed queue size in packets, where Queue size = 8 and compared our results with present case where queue size = 10kb. We received same results.

- ➔ Also see variable packet size program in Appendix: TASK 2

Results Obtained

Qmax [10,20,30,40,50], (μ) = m λ, m =[1,2,3,4] = Total 40 cases

When arrival rate (λ) = 1, m= [1:4]; Total simulated time = 1000000 sec (1.0e6), and Q_{max}

	For Mean service rate (μ) = m λ	1	2	3	4
Qmax = 10	Server utilization ρ %	90.874711 %	50.090815 %	33.307333 %	24.952303 %
	Mean number of packets in system (L)	5.001402	1.005439	0.499666	0.332313
	Mean number in the queue (L_q)	3.677544	0.504525	0.166597	0.082789
	Mean time delay in the system (W)	5.498899	1.002978	0.499100	0.332125
	Mean time in queue (Wq)	2.248539	0.334703	0.124689	0.066278
	Packets dropped	184906	6	0	0

	For Mean service rate (μ) = m λ	1	2	3	4
Qmax = 20	Server utilization ρ %	94.706984 %	50.094592 %	33.307333 %	24.952303 %
	Mean number of packets in system (L)	9.000479	1.004316	0.499666	0.332313
	Mean number in the queue (L_q)	7.600126	0.503370	0.166597	0.082789
	Mean time delay in the system (W)	9.490583	1.001815	0.499100	0.332125
	Mean time in queue (Wq)	4.241040	0.333854	0.124689	0.066278
	Packets dropped	106488	0	0	0

	For Mean service rate (μ) = m λ	1	2	3	4

	Server utilization ρ %	96.236615 %	50.094592 %	33.307333 %	24.952303 %
	Mean number of packets in system (L)	12.923580	1.004316	0.499666	0.332313
Qmax = 30	Mean number in the queue (L_q)	13.396574	0.503370	0.166597	0.082789
	Mean time delay in the system (W)	11.509513	1.001815	0.499100	0.332125
	Mean time in queue (Wq)	6.209062	0.333854	0.124689	0.066278
	Packets dropped	72406	0	0	0

	For Mean service rate (μ) = m λ	1	2	3	4
	Server utilization ρ %	97.071406 %	50.094592 %	33.307333 %	24.952303 %
	Mean number of packets in system (L)	17.063937	1.004316	0.499666	0.332313
Qmax = 40	Mean number in the queue (L_q)	15.614425	0.503370	0.166597	0.082789
	Mean time delay in the system (W)	17.542393	1.001815	0.499100	0.332125
	Mean time in queue (Wq)	8.273682	0.333854	0.124689	0.066278
	Packets dropped	57864	0	0	0

	For Mean service rate (μ) = m λ	1	2	3	4
	Server utilization ρ %	97.536393 %	50.094592 %	33.307333 %	24.952303 %
	Mean number of packets in system (L)	20.888867	1.004316	0.499666	0.332313
Qmax = 50	Mean number in the queue (L_q)	19.434777	0.503370	0.166597	0.082789

	Mean time delay in the system (W)	21.370152	1.001815	0.499100	0.332125
	Mean time in queue (Wq)	10.186488	0.333854	0.124689	0.066278
	Packets dropped	46928	0	0	0

When arrival rate (λ) = 2, m= [1:4]; Total simulated time = 1000000 sec (1.0e6), Q_{max} = 50

Qmax =10	For Mean service rate (μ) = m λ	2	4	6	8
	Server utilization ρ %	94.677536 %	50.112467%	33.347518%	24.994352 %
	Mean number of packets in system (L)	8.995821	1.004541	0.500499	0.333062
	Mean number in the queue (L_q)	7.598993	0.503416	0.167025	0.083121
	Mean time delay in the system (W)	4.745646	0.501255	0.249965	0.166394
	Mean time in queue (Wq)	2.123438	0.167187	0.062434	0.033256
	Packets dropped	211806	0	0	0

Qmax =20	For Mean service rate (μ) = m λ	2	4	6	8
	Server utilization ρ %	97.116379 %	50.112467%	33.347518 %	24.994352 %
	Mean number of packets in system (L)	17.032946	1.004541	0.500499	0.333062
	Mean number in the queue (L_q)	15.588667	0.503416	0.167025	0.083121
	Mean time delay in the system (W)	8.758759	0.501255	0.249965	0.166394
	Mean time in queue (Wq)	4.128515	0.167187	0.062434	0.033256
	Packets dropped	114498	0	0	0

Qmax =30	For Mean service rate (μ) = m λ	2	4	6	8
	Server utilization ρ %	98.016254 %	50.112467%	33.347518 %	24.994352 %
	Mean number of packets in system (L)	24.913952	1.004541	0.500499	0.333062
	Mean number in the queue (L_q)	23.457154	0.503416	0.167025	0.083121
	Mean time delay in the system (W)	12.690249	0.501255	0.249965	0.166394
	Mean time in queue (Wq)	6.096593	0.167187	0.062434	0.033256
	Packets dropped	77336	0	0	0

Qmax =40	For Mean service rate (μ) = m λ	2	4	6	8
	Server utilization ρ %	98.495365 %	50.112467%	33.347518 %	24.994352 %
	Mean number of packets in system (L)	33.264509	1.004541	0.500499	0.333062
	Mean number in the queue (L_q)	31.791483	0.503416	0.167025	0.083121
	Mean time delay in the system (W)	16.867893	0.501255	0.249965	0.166394
	Mean time in queue (Wq)	8.178337	0.167187	0.062434	0.033256
	Packets dropped	59722	0	0	0

	For Mean service rate (μ) = m λ	2	4	6	8
	Server utilization ρ %	98.788978 %	50.112467%	33.347518 %	24.994352 %
	Mean number of packets in system (L)	40.494264	1.004541	0.500499	0.333062

Qmax =50	Mean number in the queue (L_q)	39.027913	0.503416	0.167025	0.083121
	Mean time delay in the system (W)	20.472144	0.501255	0.249965	0.166394
	Mean time in queue (Wq)	9.984570	0.167187	0.062434	0.033256
	Packets dropped	47178	0	0	0

Results Discussion

We run the simulation for 1.0e6, so to remove transient conditions. As service rate increases the server utilization decreases, because the server is idle when waiting for arrival. As queue size increase, eventually less packets are dropped. Packets dropped increases with increase in arrival rate, decrease with service rate. Server utilization increases with increase in queue size.

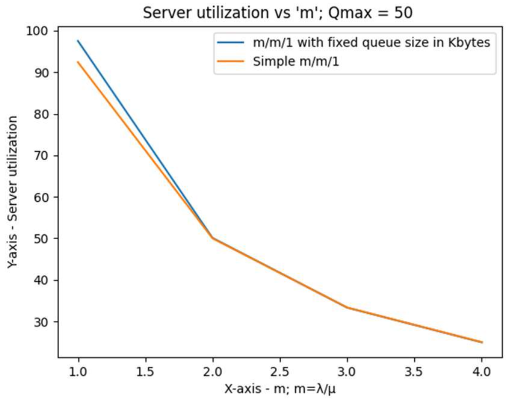
Our simulation results align with the expected analytical results which is a positive sign that our simulation shows positive correlation with expected real-world systems.

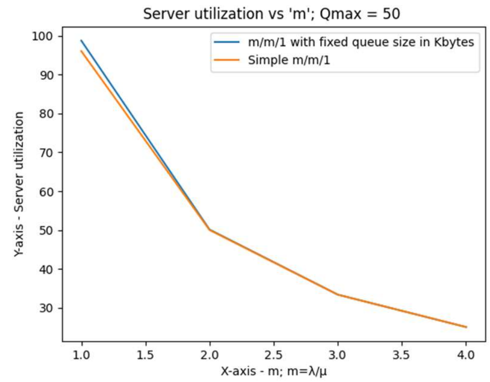
When compared to simple M/M/1, the queue size with constant or variable packets size performs better with increasing queue sizes. Even with restricted queue size, the performance and server utilization are slightly better than simple M/M/1 queueing system.

Graphs

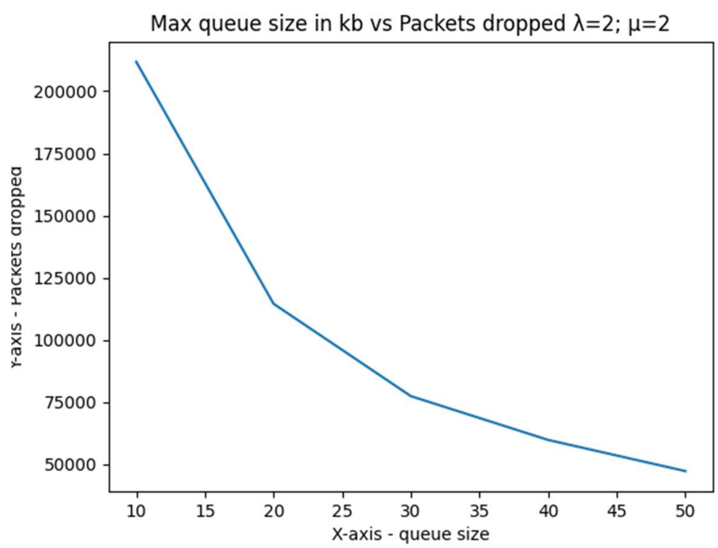
Server utilization on y-axis vs queue size on x-axis

Server utilization on y-axis vs ‘m’ on x-axis and simple MM1





Queue size in Kbytes vs drops



Appendix for all related C programs

Tip: Use notepad++ for clear interpretation

Attached programs here:

TASK 1

Simple M/M/1 queue

```
//=========================================================== file = mm1.c =====
//=  A simple "straight C" M/M/1 queue simulation                              =
//==============================================================================
//=  Notes: 1) This program is adapted from Figure 1.6 in Simulating           =
//=            Computer Systems, Techniqyes and Tools by M. H. MacDougall      =
//=            (1987).                                                         =
//=         2) The values of SIM_TIME, ARR_TIME, and SERV_TIME need to be      =
//=            set.                                                            =
//=----------------------------------------------------------------------------=
//=  Build: gcc mm1.c -lm, bcc32 mm1.c, cl mm1.c                               =
//=----------------------------------------------------------------------------=
//=  Execute: mm1                                                              =
//=----------------------------------------------------------------------------=
//=  History: KJC (03/09/99) - Genesis                                         =
//==============================================================================

//----- Include files ----------------------------------------------------------
#include <stdio.h>             // Needed for printf()
#include <stdlib.h>            // Needed for exit() and rand()
#include <math.h>              // Needed for log()

//----- Constants --------------------------------------------------------------
#define SIM_TIME   1.0e9       // Simulation time
//----- Function prototypes ----------------------------------------------------
double expntl(double x);       // Generate exponential RV with mean x

//===== Main program ===========================================================
void main(void)
{
  // tAKING USER INPUTS
  double ARR_TIME_USER;
  double SERV_TIME_USER;
  printf("Please enter Mean Arrival time: ");
  scanf("%lf", &ARR_TIME_USER);
  printf("Mean Arrival time is  %f\n",ARR_TIME_USER);
  printf("Please enter Mean Service time: ");
  scanf("%lf", &SERV_TIME_USER);
  printf("Mean Service time is %f\n",SERV_TIME_USER);
  double   end_time = SIM_TIME; // Total time to simulate
  double   Ta = ARR_TIME_USER;       // Mean time between arrivals // = 1/lambda
  double   Ts = SERV_TIME_USER;      // Mean service time // = 1/ mue

  double   time = 0.0;         // Simulation time
  double   t1 = 0.0;           // Time for next event #1 (arrival)
  double   t2 = SIM_TIME;      // Time for next event #2 (departure)
  unsigned long int n = 0;     // Number of customers in the system

  unsigned long int c = 0;     // Number of service completions
```

```
  double   b = 0.0;            // Total busy time
  double   s = 0.0;            // Area of number of customers in system
  double   tn = time;          // Variable for "last event time"
  double   tb;                 // Variable for "last start of busy time"
  double   x;                  // Throughput
  double   u;                  // Utilization
  double   l;                  // Mean number in the system
  double   w;                  // Mean residence time

  double   qs = 0.0;            // Area of number of customers in the queue
  double tq=0.0;                       //total time in queue

  // Main simulation loop
  while (time < end_time)
  {
    if (t1 < t2)                 // *** Event #1 (arrival)
    {
      time = t1;
      s = s + n * (time - tn);   // Update area under "s" curve

        if(n>1)
        {
              qs = qs + ((n-1) * (time - tn));  // Update area under "qs" curve
              //tq=tq+((n-1) * (time - tn));  //if we want the result as
W=Nq/Lamda we include this step.(with is add calculation of the queueing delay for
all customers even if they didn't leave the queue).
        }

      n++;

      tn = time;                 // tn = "last event time" for next event
      t1 = time + expntl(Ta);


      if (n == 1)
      {
        tb = time;               // Set "last start of busy time"
        t2 = time + expntl(Ts);
      }
    }
    else                         // *** Event #2 (departure)
    {
      time = t2;
      s = s + n * (time - tn);   // Update area under "s" curve

       if(n>1)
        {
              qs = qs + ((n-1) * (time - tn));
              tq=tq+((n-1) * (time - tn)); //compute the time only for the
customers that complete the times in queue (i.e who leave the queue as in slide
53).
        }

      n--;

      tn = time;                 // tn = "last event time" for next event
      c++; // Increment number of completions
```

```
      if (n > 0)
        t2 = time + expntl(Ts);
      else
      {
        t2 = SIM_TIME;
        b = b + time - tb;      // Update busy time sum if empty
      }
    }
  }

  x = c / time;   // Compute throughput rate
  u = b / time;   // Compute server utilization
  l = s / time;   // Compute mean number in system
  w = l / x;      // Compute mean residence or system time

  // Output results
  printf("============================================================= \n");
  printf("=            *** Results from M/M/1 Simulation ***            = \n");
  printf("============================================================= \n");
  printf("=  Total simulated time         = %3.4f sec  \n", end_time);
  printf("============================================================= \n");
  printf("=  INPUTS: \n");
  printf("=    Mean time between arrivals = %f sec      \n", Ta);
  printf("=    Mean service time          = %f sec      \n", Ts);
  printf("============================================================= \n");
  printf("=  OUTPUTS: \n");
  printf("=    Number of completions      = %ld cust    \n", c);
  printf("=    Throughput rate            = %f cust/sec \n", x);
  printf("=    Server utilization         = %f %%       \n", 100.0 * u);
  printf("=    Mean number in system      = %f cust     \n", l);
  printf("=    Mean residence time        = %f sec      \n", w);
  printf("=    Mean number in the queue   = %f cust      \n", qs/time);
  printf("=    Mean time in the queue     = %f sec/cust      \n", tq/(c-1));
  printf("============================================================= \n");
}
//===============================================================================
//=  Function to generate exponentially distributed RVs using inverse method  =
//=    - Input:  x (mean value of distribution)                               =
//=    - Output: Returns with exponential RV                                  =
//===============================================================================
double expntl(double x)
{
  double z;                 // Uniform random number from 0 to 1

  // Pull a uniform RV (0 < z < 1)
  do
  {
    z = ((double) rand() / RAND_MAX);
  }
  while ((z == 0) || (z == 1));

  return(-x * log(z));
}
```

Sample Output

```
Please enter Mean Arrival time: 1
Mean Arrival time is  1.000000
Please enter Mean Service time: 0.25
Mean Service time is 0.250000
```

```
================================================================
=             *** Results from M/M/1 Simulation ***            =
================================================================
=  Total simulated time         = 1000000000.0000 sec
================================================================
=  INPUTS:
=    Mean time between arrivals = 1.000000 sec
=    Mean service time          = 0.250000 sec
================================================================
=  OUTPUTS:
=    Number of completions      = 1000164866 cust
=    Throughput rate            = 1.000165 cust/sec
=    Server utilization         = 25.000554 %
=    Mean number in system      = 0.333259 cust
=    Mean residence time        = 0.333204 sec
=    Mean number in the queue   = 0.083253 cust
=    Mean time in the queue     = 0.066617 sec/cust
================================================================
--------------------------------------------------------
Please enter Mean Arrival time: 0.5
Mean Arrival time is  0.500000
Please enter Mean Service time: 0.25
Mean Service time is 0.250000
================================================================
=             *** Results from M/M/1 Simulation ***            =
================================================================
=  Total simulated time         = 1000000000.0000 sec
================================================================
=  INPUTS:
=    Mean time between arrivals = 0.500000 sec
=    Mean service time          = 0.250000 sec
================================================================
=  OUTPUTS:
=    Number of completions      = 2000293717 cust
=    Throughput rate            = 2.000294 cust/sec
=    Server utilization         = 49.999560 %
=    Mean number in system      = 0.999416 cust
=    Mean residence time        = 0.499634 sec
=    Mean number in the queue   = 0.499420 cust
=    Mean time in the queue     = 0.166518 sec/cust
================================================================
```

TASK 2

M/M/k with 2 SERVERS

```
/********************************************************************************
* Includes
*******************************************************************************/
#include <stdio.h>              // Needed for printf()
#include <stdlib.h>             // Needed for exit() and rand()
#include <unistd.h>             // Needed for getopts()
#include <stdbool.h>            // Needed for bool type

/********************************************************************************
* Defined constants and variables
* NOTE: All TIME constants are defined in seconds!
*******************************************************************************/
#define SIM_TIME   1.0e6        // Simulation time
```

```
/*****************************************************************************
* Function Prototypes
*****************************************************************************/
static void show_usage(char *name);
int min_departure(double arr[], int capacity); // Find the index of the minimum
departure time
int idle_server(double arr[], int capacity);  // Position in array to store cust.
departure time

double expntl(double x)
{
  double z;                 // Uniform random number from 0 to 1

  // Pull a uniform RV (0 < z < 1)
  do
  {
    z = ((double) rand() / RAND_MAX);
  }
  while ((z == 0) || (z == 1));

  return(-x * log(z));
}


/*****************************************************************************
* Main Function
*****************************************************************************/
int main(int argc, char **argv)
{
    // tAKING USER INPUTS
    double ARR_TIME_USER;
    double SERV_TIME_USER;
    int NUM_SERVERS = 2;
    printf("Please enter Mean Arrival time: ");
    scanf("%lf", &ARR_TIME_USER);
    printf("Mean Arrival time is  %f\n",ARR_TIME_USER);
    printf("Please enter Mean Service time: ");
    scanf("%lf", &SERV_TIME_USER);
    printf("Mean Service time is %f\n",SERV_TIME_USER);


    int opt;    // Hold the options passed as argument
    double endTime = SIM_TIME;        // Total time to do Simulation
    double arrTime = ARR_TIME_USER;        // Mean time between arrivals
    double departTime = SERV_TIME_USER;    // Mean service time
    int c = NUM_SERVERS;              // Number of servers in the system

    double time = 0.0;                  // Current Simulation time
    double nextArrival = 0.0;           // Time for next arrival
    double nextDeparture = SIM_TIME;    // Time for next departure

    int nextDepartIndex;                // Index of next departure time in array
    int arrayIndex = 0;                 // Auxiliar variable
    unsigned int n = 0;           // Actual number of customers in the system

    unsigned int departures = 0;  // Total number of customers served
    double busyTime = 0.0;        // Total busy time
    double s = 0.0;               // Area of number of customers in system
    double lastEventTime = time;  // Variable for "last event time"
```

```
double lastBusyTime;          // Variable for "last start of busy time"
double x;     // Throughput rate
double u;     // Utilization of system
double l;     // Average number of customers in system
double w;     // Average Sojourn time
double qs = 0.0;
double tq =0.0;

if (argc > 1)
{
    while ( (opt = getopt(argc, argv, "a:d:s:c:")) != -1 )
    {
        switch (opt) {
            case 'a':
                arrTime = atof(optarg);
                break;
            case 'd':
                departTime = atof(optarg);
                break;
            case 's':
                endTime = atof(optarg);
                break;
            case 'c':
                c = atoi(optarg);
                break;
            default:    // '?' unknown option
                show_usage( argv[0] );
        }
    }
}

double custDepartures[c]; // Departure times of serving customers
for (int i=0; i < c; i++)
    custDepartures[i] = SIM_TIME;   // Fill the array with maximum time

// Simulation loop
while (time < endTime)
{
    // Arrival occurred
    if (nextArrival < nextDeparture)
    {
        time = nextArrival;
        s = s + n * (time - lastEventTime);  // Update area under "s" curve

        if (n>2)
        {
            qs = qs + ((n-2) * (time - lastEventTime));
            tq=tq+((n-2) * (time - lastEventTime));
        }
        n++;    // Customers in system increase
        lastEventTime = time;   // "last event time" for next event

        nextArrival = time + expntl(arrTime);
        if (n <= c)
        {
            if (n == c)
                lastBusyTime = time;    // Set "last start of busy time"

            arrayIndex = idle_server(custDepartures, c);
```

```
                custDepartures[arrayIndex] = time + expntl(departTime);
                if (n == 1)
                {
                    nextDepartIndex = arrayIndex;
                    nextDeparture = custDepartures[nextDepartIndex];
                }
            } // end of if "n <= c"
        }
        // Departure occurred
        else
        {
            time = nextDeparture;
            s = s + n * (time - lastEventTime); // Update area under "s" curve

            if(n>2)
            {
                qs = qs + ((n-2) * (time - lastEventTime));
                tq=tq+((n-2) * (time - lastEventTime)); //compute the time only
for the customers that complete the times in queue (i.e who leave the queue as in
slide 53).
            }
            n--;    // Customers in system decrease
            lastEventTime = time;   // "last event time" for next event
            departures++;           // Increment number of completions
            custDepartures[nextDepartIndex] = SIM_TIME; // Set server as empty
            if (n > 0)
            {
                if (n == c-1) // Update busy time when at least one server idle
                    busyTime = busyTime + time - lastBusyTime;

                if (n >= c)   // Calculate departure of a waiting customer
                    custDepartures[nextDepartIndex] = time + expntl(departTime);
                // Look for the next departure time
                nextDepartIndex = min_departure(custDepartures, c);
                nextDeparture = custDepartures[nextDepartIndex];
            }
            else
                nextDeparture = SIM_TIME;
        }
    }

    // Compute outputs
    x = departures / time;  // Compute throughput rate
    u = busyTime / time;    // Compute server utilization
    l = s / time;             // Avg number of customers in the system
    w = l / x;              // Avg Sojourn time

    // Output results
    printf("<-------------------------------------------------------------> \n");
    printf("<           *** Results for M/M/%d simulation ***             > \n",
c);
    printf("<-------------------------------------------------------------> \n");
    printf("-  INPUTS: \n");
    printf("-    Total simulation time        = %.4f sec \n", endTime);
    printf("-    Mean time between arrivals   = %.4f sec \n", arrTime);
    printf("-    Mean service time            = %.4f sec \n", departTime);
    printf("-    # of Servers in system       = %d servers \n", c);
    printf("<-------------------------------------------------------------> \n");
    printf("-  OUTPUTS: \n");
```

```
    printf("-    # of Customers served        = %u cust \n", departures);
    printf("-    Throughput rate              = %f cust/sec \n", x);
    printf("-    Server utilization           = %f %% \n", 100.0 *u*2);
    printf("-    Avg # of cust. in system     = %f cust \n", l);
    printf("-    Mean Residence time          = %f sec \n", w);
    printf("-    Mean # in queue              = %f sec \n", (qs/time));
    printf("-    Mean time in queue           = %f sec \n", (tq/(departures-1)));
    printf("<-------------------------------------------------------------> \n");
}

/********************************************************************************
*       min_departure(double arr[], int capacity)
*********************************************************************************
* Function that return the index of the minimum departure time
* - Input: arr (array of departures)
* - Input: capacity (size of the array)
*******************************************************************************/
int min_departure(double arr[], int capacity)
{
    int index = 0;

    for (int i=1; i < capacity; i++)
    {
        if (arr[i] < arr[index])
            index = i;
    }
//    printf(" - INDEX = %d \n", index);
    return index;
}

/********************************************************************************
*       idle_server(double arr[], int capacity)
*********************************************************************************
* Function that return the index of an "idle server"
* It's used to determine to which position of the array we will save the
* departure time of the client (which server is serving the customer)
* - Input: arr (array of departures)
* - Input: capacity (size of the array)
*******************************************************************************/
int idle_server(double arr[], int capacity)
{
    int index = 0;
    bool idle = false;

    for (int i=1; (i < capacity && !idle); i++)
    {
        if (arr[i] == SIM_TIME)
        {
            index = i;
            idle = true;
        }
    }
//    printf(" - INDEX = %d \n", index);
    return index;
}

/********************************************************************************
*       show_usage(char *name)
*********************************************************************************
```

```
* Function that return a message of how to use this program
* - Input: name (the name of the executable)
********************************************************************************/
static void show_usage(char *name)
{
    printf("\nUsage: \n");
    printf("%s [option] value \n", name);
    printf("\n");
    printf("Options: \n");
    printf("\t-a\tMean time between arrivals (in seconds) \n");
    printf("\t-d\tMean service time (in seconds) \n");
    printf("\t-s\tTotal simulation time (in seconds) \n");
    printf("\t-c\tNumber of servers in the system\n");
    exit(EXIT_SUCCESS);
}
```

Sample Output

```
Please enter Mean Arrival time: 0.5
Mean Arrival time is  0.500000
Please enter Mean Service time: 0.5
Mean Service time is 0.500000
<-------------------------------------------------------------->
<            *** Results for M/M/2 simulation ***              >
<-------------------------------------------------------------->
-  INPUTS:
-    Total simulation time        = 1000000.0000 sec
-    Mean time between arrivals   = 0.5000 sec
-    Mean service time            = 0.5000 sec
-    # of Servers in system       = 2 servers
<-------------------------------------------------------------->
-  OUTPUTS:
-    # of Customers served        = 2002946 cust
-    Throughput rate              = 2.002946 cust/sec
-    Server utilization           = 69.269680 %
-    Avg # of cust. in system     = 1.333644 cust
-    Mean Residence time          = 0.665841 sec
-    Mean # in queue              = 0.332547 sec
-    Mean time in queue           = 0.166029 sec
<-------------------------------------------------------------->
```

M/M/1 with Fixed Queue size in no. of arrivals

```
//=========================================================== file = mm1.c =====
//=  A simple "straight C" M/M/1 queue simulation                              =
//==============================================================================
//=  Notes: 1) This program is adapted from Figure 1.6 in Simulating           =
//=            Computer Systems, Techniqyes and Tools by M. H. MacDougall      =
//=            (1987).                                                         =
//=         2) The values of SIM_TIME, ARR_TIME, and SERV_TIME need to be      =
//=            set.                                                            =
//=----------------------------------------------------------------------------=
//=  Build: gcc mm1.c -lm, bcc32 mm1.c, cl mm1.c                               =
//=----------------------------------------------------------------------------=
//=  Execute: mm1                                                              =
//=----------------------------------------------------------------------------=
//=  Perfected by: github.com/mashahzad                                        =
//=----------------------------------------------------------------------------=
```

```
//=  History: KJC (03/09/99) - Genesis                                       =
//===========================================================================

//----- Include files -------------------------------------------------------
#include <stdio.h>              // Needed for printf()
#include <stdlib.h>             // Needed for exit() and rand()
#include <math.h>               // Needed for log()

//----- Constants -----------------------------------------------------------
#define SIM_TIME   1.0e6        // Simulation time

//----- Function prototypes -------------------------------------------------
double expntl(double x);        // Generate exponential RV with mean x

//===== Main program ========================================================
void main(void)
{
  // tAKING USER INPUTS
  double ARR_TIME_USER;
  double SERV_TIME_USER;
  int QUEUE_MAX_USER;
  printf("Please enter Mean Arrival time: ");
  scanf("%lf", &ARR_TIME_USER);
  printf("Mean Arrival time is  %f\n",ARR_TIME_USER);
  printf("Please enter Mean Service time: ");
  scanf("%lf", &SERV_TIME_USER);
  printf("Mean Service time is %f\n",SERV_TIME_USER);
  printf("Please enter Max Queue size: ");
  scanf("%d", &QUEUE_MAX_USER);
  printf("Max Queue size is %d\n",QUEUE_MAX_USER);

  double   end_time = SIM_TIME; // Total time to simulate
  double   Ta = ARR_TIME_USER;       // Mean time between arrivals
  double   Ts = SERV_TIME_USER;      // Mean service time

  double   time = 0.0;          // Simulation time
  double   t1 = 0.0;            // Time for next event #1 (arrival)
  double   t2 = SIM_TIME;       // Time for next event #2 (departure)
  unsigned long int n = 0;      // Number of customers in the system

  unsigned long int c = 0;      // Number of service completions
  double   b = 0.0;             // Total busy time
  double   s = 0.0;             // Area of number of customers in system
  double   tn = time;           // Variable for "last event time"
  double   tb;                  // Variable for "last start of busy time"
  double   x;                   // Throughput
  double   u;                   // Utilization
  double   l;                   // Mean number in the system
  double   w;                   // Mean residence time
  double   qs = 0.0;             // Area of number of customers in the queue
  double tq=0.0;                        //total time in queue
  unsigned long int queue = 0;
  int drop = 0;

  // Main simulation loop
  while (time < end_time)
  {
    if (t1 < t2)                  // *** Event #1 (arrival)
    {
```

```
      time = t1;
      s = s + n * (time - tn);   // Update area under "s" curve

        if(n>1)
        {
            if(n <= QUEUE_MAX_USER+1) //total cust in system = cusomers in queue
+ 1 in service
           {
               queue = (n-1); //number in queue = Number in system minus one in
service
               qs = qs + ((n-1) * (time - tn));  // Update area under "qs" curve
             //tq=tq+((n-1) * (time - tn));  //if we want the result as
W=Nq/Lamda we include this step.(with is add calculation of the queueing delay for
all customers even if they didn't leave the queue).
           }
           else
           {
               drop = drop + (n - QUEUE_MAX_USER);
               n = QUEUE_MAX_USER+1; //drop excess customers

           }
        }

      n++;

      tn = time;                 // tn = "last event time" for next event
      t1 = time + expntl(Ta);


      if (n == 1)
      {
        tb = time;               // Set "last start of busy time"
        t2 = time + expntl(Ts);
      }
    }
    else                         // *** Event #2 (departure)
    {
      time = t2;
      s = s + n * (time - tn);   // Update area under "s" curve

       if(n>1)
        {
             qs = qs + ((n-1) * (time - tn));
             tq=tq+((n-1) * (time - tn)); //compute the time only for the
customers that complete the times in queue (i.e who leave the queue as in slide
53).
        }

      n--;

      tn = time;                 // tn = "last event time" for next event
      c++; // Increment number of completions


      if (n > 0)
        t2 = time + expntl(Ts);
      else
      {
        t2 = SIM_TIME;
```

```
        b = b + time - tb;      // Update busy time sum if empty
      }
    }
  }

  x = c / time;   // Compute throughput rate
  u = b / time;   // Compute server utilization
  l = s / time;   // Compute mean number in system
  w = l / x;      // Compute mean residence or system time

  // Output results
  printf("=============================================================== \n");
  printf("=            *** Results from M/M/1 Simulation ***            = \n");
  printf("=============================================================== \n");
  printf("=  Total simulated time         = %3.4f sec  \n", end_time);
  printf("=============================================================== \n");
  printf("=  INPUTS: \n");
  printf("=    Mean time between arrivals = %f sec      \n", Ta);
  printf("=    Mean service time          = %f sec      \n", Ts);
  printf("=    Maximum Queue Size         = %d cust      \n", QUEUE_MAX_USER);
  printf("=============================================================== \n");
  printf("=  OUTPUTS: \n");
  printf("=    Number of completions      = %ld cust    \n", c);
  printf("=    Throughput rate            = %f cust/sec \n", x);
  printf("=    Server utilization         = %f %%       \n", 100.0 * u);
  printf("=    Mean number in system      = %f cust     \n", l);
  printf("=    Mean residence time        = %f sec      \n", w);
  printf("=    Mean number in the queue   = %f cust      \n", qs/time);
  printf("=    Mean time in the queue     = %f sec/cust      \n", tq/(c-1));
  printf("=    Customers dropped          = %d cust     \n",drop);
  printf("=============================================================== \n");
}

//===============================================================================
//=  Function to generate exponentially distributed RVs using inverse method  =
//=    - Input:  x (mean value of distribution)                                =
//=    - Output: Returns with exponential RV                                   =
//===============================================================================
double expntl(double x)
{
  double z;                 // Uniform random number from 0 to 1

  // Pull a uniform RV (0 < z < 1)
  do
  {
    z = ((double) rand() / RAND_MAX);
  }
  while ((z == 0) || (z == 1));

  return(-x * log(z));
}
```

Sample Output

```
Please enter Mean Arrival time: 1
Mean Arrival time is  1.000000
Please enter Mean Service time: 1
Mean Service time is 1.000000
Please enter Max Queue size: 30
```

```
Max Queue size is 30
===============================================================
=            *** Results from M/M/1 Simulation ***            =
===============================================================
=  Total simulated time         = 1000000.0000 sec
===============================================================
=  INPUTS:
=    Mean time between arrivals = 1.000000 sec
=    Mean service time          = 1.000000 sec
=    Maximum Queue Size         = 30 cust
===============================================================
=  OUTPUTS:
=    Number of completions      = 970645 cust
=    Throughput rate            = 0.970644 cust/sec
=    Server utilization         = 96.874007 %
=    Mean number in system      = 15.925432 cust
=    Mean residence time        = 16.407084 sec
=    Mean number in the queue   = 14.487437 cust
=    Mean time in the queue     = 7.716019 sec/cust
=    Customers dropped          = 60822 cust
===============================================================
```

M/M/1 with Fixed Queue size in Kbytes

```
//========================================================= file = mm1.c =====
//=  A simple "straight C" M/M/1 queue simulation                             =
//=============================================================================
//=  Notes: 1) This program is adapted from Figure 1.6 in Simulating          =
//=            Computer Systems, Techniqyes and Tools by M. H. MacDougall     =
//=            (1987).                                                        =
//=         2) The values of SIM_TIME, ARR_TIME, and SERV_TIME need to be     =
//=            set.                                                           =
//=---------------------------------------------------------------------------=
//=  Build: gcc mm1.c -lm, bcc32 mm1.c, cl mm1.c                              =
//=---------------------------------------------------------------------------=
//=  Execute: mm1                                                             =
//=---------------------------------------------------------------------------=
//=  Perfected by: github.com/mashahzad                                       =
//=---------------------------------------------------------------------------=
//=  History: KJC (03/09/99) - Genesis                                        =
//=============================================================================

//----- Include files ---------------------------------------------------------
#include <stdio.h>              // Needed for printf()
#include <stdlib.h>             // Needed for exit() and rand()
#include <math.h>               // Needed for log()

//----- Constants -------------------------------------------------------------
#define SIM_TIME   1.0e6        // Simulation time
//#define TRANSMISSION_RATE 1.0e7  // R = 10mbps
#define TRANSMISSION_RATE 1.0e4  // R = 10kbps

//----- Function prototypes ---------------------------------------------------
double expntl(double x);        // Generate exponential RV with mean x

//===== Main program ==========================================================
void main(void)
```

```
{
  // tAKING USER INPUTS
  double ARR_TIME_USER;
  double SERV_TIME_USER;
  double QUEUE_MAX_USER;
  printf("Please enter Mean Arrival time: ");
  scanf("%lf", &ARR_TIME_USER);
  printf("Mean Arrival time is  %f sec\n",ARR_TIME_USER);
  printf("Please enter Mean Service time: ");
  scanf("%lf", &SERV_TIME_USER);
  printf("Mean Service time is %f sec\n",SERV_TIME_USER);
  printf("Please enter Max Queue size in Kbytes: ");
  scanf("%lf", &QUEUE_MAX_USER);
  printf("Max Queue size is %.2f KBytes\n",QUEUE_MAX_USER);

  double   end_time = SIM_TIME; // Total time to simulate
  double   Ta = ARR_TIME_USER;       // Mean time between arrivals
  double   Ts = SERV_TIME_USER;      // Mean service time

  double   time = 0.0;         // Simulation time
  double   t1 = 0.0;           // Time for next event #1 (arrival)
  double   t2 = SIM_TIME;      // Time for next event #2 (departure)
  double n = 0.0;              // Data of Packets in system in kbytes

  unsigned long int c = 0;     // Number of service completions
  double   b = 0.0;            // Total busy time
  double   s = 0.0;            // Area of number of customers in system
  double   tn = time;          // Variable for "last event time"
  double   tb;                 // Variable for "last start of busy time"
  double   x;                  // Throughput
  double   u;                  // Utilization
  double   l;                  // Mean number in the system
  double   w;                  // Mean residence time

  double   qs = 0.0;            // Area of number of customers in the queue
  double tq=0.0;                       //total time in queue
  double queue = 0.0;
  double drop = 0; // in bytes
  double transmission_rate_Bps = TRANSMISSION_RATE/8; // = x bytes per second
  double packet_size_bytes = SERV_TIME_USER*transmission_rate_Bps; // = bytes per
packet
  double packet_size_Kbytes = packet_size_bytes/1000; // = Kilo bytes per packet
  double queue_size = QUEUE_MAX_USER/packet_size_Kbytes; // queue size limit for
given transmission and packet size (how much data fills queue)

  // Main simulation loop
  while (time < end_time)
  {
    if (t1 < t2)                 // *** Event #1 (arrival)
    {
      time = t1;
      //s = s + (n * (time - tn)/packet_size_Kbytes);   // Update area under "s"
curve
      //n_no = (n/packet_size_Kbytes);
      s = s + ( n * (time - tn)); // Update area under "s" curve

         if(n > 1)
         {
```

```
            if(n <= (queue_size+1)) //total packets data in system = cusomers in
queue + 1 in service
            {
                //queue = ((n-packet_size_Kbytes)/packet_size_Kbytes); //number in
queue = Number in system minus one in service
                qs = qs + (n-1) * (time - tn);  // Update area under "qs" curve
              //tq=tq+((n-1) * (time - tn));  //if we want the result as
W=Nq/Lamda we include this step.(with is add calculation of the queueing delay for
all customers even if they didn't leave the queue).
            }
            else
            {
                drop = drop + (n - queue_size); //data packets dropped
                n = (queue_size+1); //drop excess packets

            }
         }

      //n++;
      n++;

      tn = time;                 // tn = "last event time" for next event
      t1 = time + expntl(Ta);


      if (n == 1)
      {
        tb = time;               // Set "last start of busy time"
        t2 = time + expntl(Ts);
      }
    }
    else                         // *** Event #2 (departure)
    {
      time = t2;
      //s = s + (n * (time - tn)/packet_size_Kbytes);   // Update area under "s"
curve
      s = s + (n * (time - tn));

        if(n > 1)
         {
              //qs = qs + ((n-packet_size_Kbytes) * (time - tn));
              qs = qs + ((n-1) * (time - tn));  // Update area under "qs" curve
              tq = tq +(n-1) * (time - tn); //compute the time only for the
customers that complete the times in queue (i.e who leave the queue as in slide
53).
         }

      n--;

      tn = time;                 // tn = "last event time" for next event
      c++; // Increment number of completions


      if (n > 0)
        t2 = time + expntl(Ts);
      else
      {
        t2 = SIM_TIME;
        b = b + time - tb;      // Update busy time sum if empty
```

```
      }
    }
  }

  x = c / time;   // Compute throughput rate
  u = b / time;   // Compute server utilization
  l = s / time;   // Compute mean number in system
  w = l / x;      // Compute mean residence or system time

  // Output results
  printf("============================================================= \n");
  printf("=            *** Results from M/M/1 Simulation ***            = \n");
  printf("============================================================= \n");
  printf("=  Total simulated time         = %3.4f sec  \n", end_time);
  printf("============================================================= \n");
  printf("=  INPUTS: \n");
  printf("=    Mean time between arrivals = %f sec      \n", Ta);
  printf("=    Mean service time          = %f sec      \n", Ts);
  printf("=    Transmission Rate R        = %.2f bits per sec     \n",
TRANSMISSION_RATE);
  printf("=    Transmission Rate R        = %.2f Bytes per sec     \n",
TRANSMISSION_RATE/8);
  printf("=    Maximum Queue Size Kbytes  = %.2f Kbytes     \n", QUEUE_MAX_USER);
  printf("=    Following output when packet size is constant at %f KBytes \n",
packet_size_Kbytes);
  printf("============================================================= \n");
  printf("=  OUTPUTS: \n");
  printf("=    Number of completions      = %d packets    \n", c);
  printf("=    Throughput rate            = %f packet/sec \n", x);
  printf("=    Server utilization         = %f %%       \n", 100*u);
  printf("=    Mean number in system      = %f packet     \n", l);
  printf("=    Mean residence time        = %f sec      \n", w);
  printf("=    Mean number in the queue   = %f packet      \n", qs/time);
  printf("=    Mean time in the queue     = %f sec/packet      \n", tq/(c-1));
  printf("=    Packets dropped            = %.0f packets     \n",drop);
  printf("============================================================= \n");
}

//===========================================================================
//=  Function to generate exponentially distributed RVs using inverse method  =
//=    - Input:  x (mean value of distribution)                             =
//=    - Output: Returns with exponential RV                                =
//===========================================================================
double expntl(double x)
{
  double z;                 // Uniform random number from 0 to 1

  // Pull a uniform RV (0 < z < 1)
  do
  {
    z = ((double) rand() / RAND_MAX);
  }
  while ((z == 0) || (z == 1));

  return(-x * log(z));
}
```

Sample Output

```
Please enter Mean Arrival time: 1
Mean Arrival time is  1.000000 sec
Please enter Mean Service time: 1
Mean Service time is 1.000000 sec
Please enter Max Queue size in Kbytes: 50
Max Queue size is 50.00 KBytes
===============================================================
=            *** Results from M/M/1 Simulation ***            =
===============================================================
=  Total simulated time         = 1000000.0000 sec
===============================================================
=  INPUTS:
=    Mean time between arrivals = 1.000000 sec
=    Mean service time          = 1.000000 sec
=    Transmission Rate R        = 10000.00 bits per sec
=    Transmission Rate R        = 1250.00 Bytes per sec
=    Maximum Queue Size Kbytes  = 50.00 Kbytes
=    Following output when packet size is constant at 1.250000 KBytes
===============================================================
=  OUTPUTS:
=    Number of completions      = 977481 packets
=    Throughput rate            = 0.977479 packet/sec
=    Server utilization         = 97.536393 %
=    Mean number in system      = 20.888867 packet
=    Mean residence time        = 21.370152 sec
=    Mean number in the queue   = 19.434777 packet
=    Mean time in the queue     = 10.186488 sec/packet
=    Packets dropped            = 46928 packets
===============================================================
```

M/M/1 with Fixed Queue size with variable packet size in Kbytes

```
//========================================================== file = mm1.c =====
//=  A simple "straight C" M/M/1 queue simulation                             =
//=============================================================================
//=  Notes: 1) This program is adapted from Figure 1.6 in Simulating          =
//=            Computer Systems, Techniqyes and Tools by M. H. MacDougall     =
//=            (1987).                                                        =
//=         2) The values of SIM_TIME, ARR_TIME, and SERV_TIME need to be     =
//=            set.                                                           =
//=---------------------------------------------------------------------------=
//=  Build: gcc mm1.c -lm, bcc32 mm1.c, cl mm1.c                              =
//=---------------------------------------------------------------------------=
//=  Execute: mm1                                                             =
//=---------------------------------------------------------------------------=
//=  Perfected by: github.com/mashahzad                                       =
//=---------------------------------------------------------------------------=
//=  History: KJC (03/09/99) - Genesis                                        =
//=============================================================================

//----- Include files ---------------------------------------------------------
#include <stdio.h>             // Needed for printf()
#include <stdlib.h>            // Needed for exit() and rand()
#include <math.h>              // Needed for log()

//----- Constants -------------------------------------------------------------
```

```
#define SIM_TIME   1.0e6       // Simulation time
//#define TRANSMISSION_RATE 1.0e7  // R = 10mbps
#define TRANSMISSION_RATE 1.0e4  // R = 10kbps

//----- Function prototypes -------------------------------------------------
double expntl(double x);       // Generate exponential RV with mean x

//===== Main program ==========================================================
void main(void)
{
  // tAKING USER INPUTS
  double ARR_TIME_USER;
  double SERV_TIME_USER;
  double QUEUE_MAX_USER;
  printf("Please enter Mean Arrival time: ");
  scanf("%lf", &ARR_TIME_USER);
  printf("Mean Arrival time is  %f sec\n",ARR_TIME_USER);
  printf("Please enter Mean Service time: ");
  scanf("%lf", &SERV_TIME_USER);
  printf("Mean Service time is %f sec\n",SERV_TIME_USER);
  printf("Please enter Max Queue size in Kbytes: ");
  scanf("%lf", &QUEUE_MAX_USER);
  printf("Max Queue size is %.2f KBytes\n",QUEUE_MAX_USER);

  double   end_time = SIM_TIME; // Total time to simulate
  double   Ta = ARR_TIME_USER;       // Mean time between arrivals
  double   Ts = SERV_TIME_USER;      // Mean service time

  double   time = 0.0;         // Simulation time
  double   t1 = 0.0;           // Time for next event #1 (arrival)
  double   t2 = SIM_TIME;      // Time for next event #2 (departure)
  double n = 0.0;              // Data of Packets in system in kbytes

  unsigned long int c = 0;     // Number of service completions
  double   b = 0.0;            // Total busy time
  double   s = 0.0;            // Area of number of customers in system
  double   tn = time;          // Variable for "last event time"
  double   tb;                 // Variable for "last start of busy time"
  double   x;                  // Throughput
  double   u;                  // Utilization
  double   l;                  // Mean number in the system
  double   w;                  // Mean residence time

  double   qs = 0.0;            // Area of number of customers in the queue
  double tq=0.0;                       //total time in queue
  double queue = 0.0;
  double drop = 0; // in bytes
  double transmission_rate_Bps = TRANSMISSION_RATE/8; // = x bytes per second
  double packet_size_bytes = SERV_TIME_USER*transmission_rate_Bps; // = bytes per
packet
  double packet_size_Kbytes = packet_size_bytes/1000; // = Kilo bytes per packet
  double queue_size = QUEUE_MAX_USER; // queue size limit for given transmission
and packet size (how much data fills queue)
  double queue_data = 0.0;
  double total_data =0.0;

  // Main simulation loop
  while (time < end_time)
  {
```

```
    if (t1 < t2)                 // *** Event #1 (arrival)
    {
      time = t1;
      //s = s + (n * (time - tn)/packet_size_Kbytes);   // Update area under "s"
curve
      s = s + ( n * (time - tn)); // Update area under "s" curve

         if(n > 1)
         {
             if(n <= (queue_size/packet_size_Kbytes)+1 && queue_data<=queue_size)
//total packets data in system = cusomers in queue + 1 in service
            {
                total_data = total_data + queue_data;
                //queue = ((n-packet_size_Kbytes)/packet_size_Kbytes); //number in
queue = Number in system minus one in service
                qs = qs + (n-1) * (time - tn);  // Update area under "qs" curve
              //tq=tq+((n-1) * (time - tn));  //if we want the result as
W=Nq/Lamda we include this step.(with is add calculation of the queueing delay for
all customers even if they didn't leave the queue).
            }
            else
            {
                drop = drop + (n - queue_size/packet_size_Kbytes); //data packets
dropped
                n = ((queue_size/packet_size_Kbytes)+1); //drop excess packets
                queue_data = queue_size;
                total_data = total_data + queue_data;

            }
         }

      //n++;
      n++;
      queue_data = queue_data + (packet_size_Kbytes + expntl(Ta/1000)); //
Generating random sized random packets for processing
      total_data = total_data + queue_data;
      // ==========UNCOMMENT BELOW LINES AND SET SIM_TIME TO 1.0e2 FOR OBSERVING
VALUES=========
      //printf("Queue data at # %f is %f:::",n,queue_data);
      //printf("Elements in queue = %f \n", n-1);

      tn = time;                 // tn = "last event time" for next event
      t1 = time + expntl(Ta);


      if (n == 1)
      {
        tb = time;               // Set "last start of busy time"
        t2 = time + expntl(Ts);
      }
    }
    else                         // *** Event #2 (departure)
    {
      time = t2;
      //s = s + (n * (time - tn)/packet_size_Kbytes);   // Update area under "s"
curve
      s = s + (n * (time - tn));

        if(n > 1)
```

```
        {
              //qs = qs + ((n-packet_size_Kbytes) * (time - tn));
              qs = qs + ((n-1) * (time - tn));  // Update area under "qs" curve
              tq = tq +(n-1) * (time - tn); //compute the time only for the
customers that complete the times in queue (i.e who leave the queue as in slide
53).
        }

      n--;
      queue_data = queue_data - 1.25;

      tn = time;                 // tn = "last event time" for next event
      c++; // Increment number of completions


      if (n > 0)
        t2 = time + expntl(Ts);
      else
      {
        t2 = SIM_TIME;
        b = b + time - tb;      // Update busy time sum if empty
      }
    }
  }

  x = c / time;   // Compute throughput rate
  u = b / time;   // Compute server utilization
  l = s / time;   // Compute mean number in system
  w = l / x;      // Compute mean residence or system time

  // Output results
  printf("=============================================================== \n");
  printf("=            *** Results from M/M/1 Simulation ***            = \n");
  printf("=============================================================== \n");
  printf("=  Total simulated time         = %3.4f sec  \n", end_time);
  printf("=============================================================== \n");
  printf("=  INPUTS: \n");
  printf("=    Mean time between arrivals = %f sec      \n", Ta);
  printf("=    Mean service time          = %f sec      \n", Ts);
  printf("=    Transmission Rate R        = %.2f bits per sec     \n",
TRANSMISSION_RATE);
  printf("=    Transmission Rate R        = %.2f Bytes per sec     \n",
TRANSMISSION_RATE/8);
  printf("=    Maximum Queue Size Kbytes  = %.2f Kbytes     \n", QUEUE_MAX_USER);
  printf("=    Following output when packet size is variable \n");
  printf("=============================================================== \n");
  printf("=  OUTPUTS: \n");
  printf("=    Number of completions      = %d packets    \n", c);
  printf("=    Throughput rate            = %f packet/sec \n", x);
  printf("=    Server utilization         = %f %%       \n", 100*u);
  printf("=    Mean number in system      = %f packet     \n", l);
  printf("=    Mean residence time        = %f sec      \n", w);
  printf("=    Mean number in the queue   = %f packet      \n", qs/time);
  printf("=    Mean time in the queue     = %f sec/packet      \n", tq/(c-1));
  printf("=    Packets dropped            = %.0f packets     \n",drop);
  printf("=============================================================== \n");
}

//=================================================================
```

```
//=  Function to generate exponentially distributed RVs using inverse method  =
//=    - Input:  x (mean value of distribution)                                =
//=    - Output: Returns with exponential RV                                   =
//=============================================================================
double expntl(double x)
{
  double z;                 // Uniform random number from 0 to 1

  // Pull a uniform RV (0 < z < 1)
  do
  {
    z = ((double) rand() / RAND_MAX);
  }
  while ((z == 0) || (z == 1));

  return(-x * log(z));
}
```

Sample Output

```
Please enter Mean Arrival time: 0.5
Mean Arrival time is  0.500000 sec
Please enter Mean Service time: 0.25
Mean Service time is 0.250000 sec
Please enter Max Queue size in Kbytes: 20
Max Queue size is 20.00 KBytes
===============================================================
=             *** Results from M/M/1 Simulation ***           =
===============================================================
=  Total simulated time         = 1000000.0000 sec
===============================================================
=  INPUTS:
=    Mean time between arrivals = 0.500000 sec
=    Mean service time          = 0.250000 sec
=    Transmission Rate R        = 10000.00 bits per sec
=    Transmission Rate R        = 1250.00 Bytes per sec
=    Maximum Queue Size Kbytes  = 20.00 Kbytes
=    Following output when packet size is variable
===============================================================
=  OUTPUTS:
=    Number of completions      = 2004055 packets
=    Throughput rate            = 2.004055 packet/sec
=    Server utilization         = 50.112467 %
=    Mean number in system      = 1.004541 packet
=    Mean residence time        = 0.501255 sec
=    Mean number in the queue   = 0.503416 packet
=    Mean time in the queue     = 0.167187 sec/packet
=    Packets dropped            = 0 packets
===============================================================
```

References

[1] Ghimire, Sushil & Thapa, Gyan & Ghimire, R.P & Silvestrov, Sergei. (2017). A Survey on Queueing Systems with Mathematical Models and Applications. American Journal of Operational Research. 2017. 1-14. 10.5923/j.ajor.20170701.01.

https://www.researchgate.net/publication/321134920_A_Survey_on_Queueing_Systems_with_Mathematical_Models_and_Applications

[2] Queuing Theory calculator https://queue-theory-calculator.vercel.app/

[3] M/M/2 queuing system https://www.johndcook.com/blog/2022/01/12/mm2/

[4] *The Art of Computer Systems Performance Analysis, by Raj Jain* Wiley, New York, NY 10158-0012, ISBN: 0471-50336-3, April 1991, 720 pp.

Part VI: Queueing Models

***