Department of Computer Science BOSTON UNIVERSITY

CS-350 - Fundamentals of Computing Systems

Midterm Exam #1
Fall 2024

Name: ______________________________

BU Username: ____________________ BU ID: ____________________

NOTE: *Please use only the provided space and the included extra page to answer the questions. If you do use the extra pages, make sure you reference them properly in your solutions. The very last page can be detached for your convenience.*

Problem #1:	/14
Problem #2:	/25
Problem #3:	/22
Problem #4:	/22
Problem #5:	/25
Total Score:	/108

Remarks:

- This is a closed-book/notes exam.
- Basic calculators are allowed.
- You have 80 minutes to complete your exam.
- There are 108 total points.
- If your score is more than 100, it is capped at 100.
- Show your work for full marks.
- Problems and sub-problems weighted as shown.
- **Explain all your assumptions clearly.**

A1B2C3D4E5F6G7H8I9J0

Problem 1

Label each of the statements below with either **True (T)** or **False (F)**:

	Statement	**T/F**
a.	If two systems are characterized by the same availability, then they can be considered more predictable that two systems with different availability.	**FALSE**
b.	An M/M/1/K queuing system is always able to reach steady-state operation regardless of the traffic input rate λ.	**TRUE**
c.	Increasing the multi-programming level (MPL) in a system always leads to an increase in the capacity of the system.	**FALSE**
d.	Consider two different systems, namely A and B. Even if the capacity of B is twice that of A, it is still possible for B to have a throughput lower than A.	**TRUE**
e.	If a process is blocked from the perspective of a given resource, then this process is not currently contributing to the resource utilization.	**TRUE**
f.	The `pthread_create(...)` function returns only when the thread spawned by the function terminates its execution and exits.	**FALSE**
g.	The steady-state expected response time in an M/M/1 system is always N times higher compared to an N*M/M/1 system subject to the same input traffic rate λ and identical per-server mean service time T_s.	**FALSE**
h.	Assume that Poisson-distributed events of type A and B arrive with a mean rate of λ_A and λ_B, respectively. Then, the mean time that elapses between any two events, regardless of their type, is $\frac{1}{\lambda_A+\lambda_B}$	**TRUE**
i.	If a system is operating close to its capacity, a very small input traffic increase might lead to a very large increase in the end-to-end response time.	**TRUE**
j.	Just like an M/M/1 system, an M/D/1 system will be unable to reach steady state if the input traffic rate approaches the system's service rate.	**TRUE**

Note: There are 10 questions. A correct answer will get you 2 points; an incorrect answer -1 points; a blank answer 0 points. The final score is capped at 14.

A1B2C3D4E5F6G7H8I9J0

Problem 2

ChatGPT generated the following incomplete code of the `handle_connection(...)` implementation for a server that uses a worker child thread to process transactions via a FIFO queue shared with the parent thread. Elements in the queue are added and popped via the `add_to_queue(...)` and `get_from_queue(...)` functions, respectively. When handling a request, the worker thread busywaits for the length of the request, and then responds to the client with the same request ID.

```c
struct request {
  uint64_t req_id; /* Integer request ID set by the client */
  struct timespec req_timestamp; /* time when request sent by client   */
  struct timespec req_length; /* time length of the request */
};
struct request_meta {
  struct request request;
  struct timespec receipt_timestamp; /* time when request enqueued */
  struct timespec start_timestamp;   /* time when request starts service */
  struct timespec completion_timestamp; /* time when request completed */
};
struct worker_params {
  int conn_socket;
  struct queue * the_queue;
};
int worker_main (void * arg) { /* Main logic of the worker thread */
  struct timespec now;
  struct worker_params * params = (struct worker_params *)arg;
  while (/* Worker termination condition -- CODE OMITTED */) {
    struct request_meta req;
    struct response resp;
    req = get_from_queue(params->the_queue);
    clock_gettime(CLOCK_MONOTONIC, &req.start_timestamp);
    busywait_timespec(req.completion_timestamp);
    clock_gettime(CLOCK_MONOTONIC, &req.request.req_length);
    resp.req_id = req.request.req_id;
    send(params->conn_socket, &resp, sizeof(struct response), 0);
    printout_completed_request(req);
    dump_queue_status(params->the_queue);
  }
  return EXIT_SUCCESS;
}
void handle_connection(int conn_socket) { /* Main connection handling logic */
  struct queue * the_queue;
  int in_bytes, worker_id;
  struct request_meta * req = (struct request_meta *)malloc(sizeof(struct request_meta));
  the_queue = initialize_queue(QUEUE_SIZE);
  worker_id = start_worker(conn_socket, the_queue);
  do {
    in_bytes = recv(conn_socket, &req->request, sizeof(struct request), 0);
    clock_gettime(CLOCK_MONOTONIC, &req->receipt_timestamp);
    if (in_bytes > 0) {
      add_to_queue(*req, the_queue);
    }
  } while (in_bytes > 0);
  /* Terminate worker and shutdown socket -- CODE OMITTED */
}
```

Listing 1: Handle connection and worker implementation

A1B2C3D4E5F6G7H8I9J0

(a) **[6 points]** The code seems to compile without problems, but when you execute it and try to compute the statistics for your EVAL assignment from it output, something seems very off. Sometimes, the server just stalls forever while processing some requests. Moreover, when attempting to plot the service time distribution, the resulting distribution is unrecognizable. Identify the cause of the bug. You can directly refer to the line number of the code line(s) that are problematic. Do your best to explain the behavior of (1) long stalls upon processing some requests, and (2) unexpected service time distribution.

Solution:
There are two problematic lines, namely Line 24 and Line 25.

Line 24 incorrectly uses the request completion timestamp as the amount of time to busywait for.

Problem (1): since the completion timestamp is generally uninitialized, if the (uninitialized) timestamp value happens to be a very large number, the busywait might last for a long time, making the server appear stuck on processing a given request.

Line 25 incorrectly overwrites the request length sent by the client with the current timestamp at the time of request completion.

Problem (2): since the completion time is never correctly produced in output, computing the service time as (completion time - start time) yields incorrect results. Moreover, since the request lengths are being overwritten, using the request lengths in output does not produce any recognizable distribution.

(b) **[6 points]** Provide a fix for the code. For each problematic line identified above, write down how the line should read instead. Make sure to indicate which line (identified by its line number) in the original code your line will replace.

Solution:
The following two lines will be replaced:

Line 24 with: `busywait_timespec (req.request.req_length);`

Line 25 with: `clock_gettime(CLOCK_MONOTONIC, &req.completion_timestamp);`

A1B2C3D4E5F6G7H8I9J0

(c) **[6 points]** After running a long experiment and post-processing the server's output, you have produced the following plots for the request inter-arrival times (Figure 1) and service times (Figure 2). Apart from the distributions of the experimental data, some theoretical distributions have been added to the plots for your convenience.

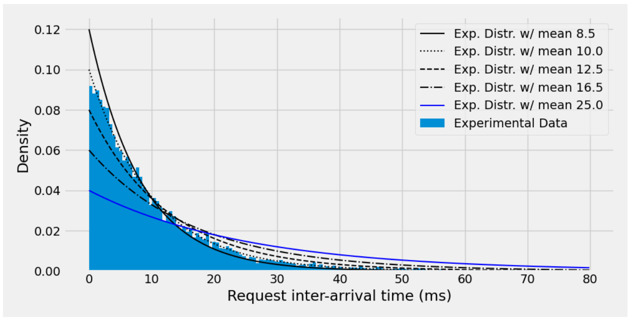

Figure 1: Experimental and theoretical inter-arrival time densities.

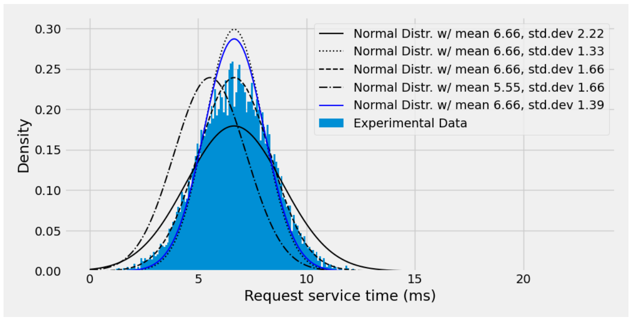

Figure 2: Experimental and theoretical service time densities.

What is the expected utilization of the server at the end of the experiment?

Solution:

The curve that best match the distribtuion of inter-arrival times is an exponential distribution with mean $\frac{1}{lambda} = 10.0$ ms, thus with a mean arrival rate of $\lambda = 0.1$ req./ms.

The curve that best match the distribtuion of service times is a normal distribution with mean $T_s = 6.66$ and standard deviation $\sigma_{T_s} = 1.66$.

Thus, the expected utilization of the server is $\rho = \lambda T_s = 0.67$

(d) **[7 points]** Considering the same experimental data provided in Figure 1 and Figure 2, what is the expected response time for a generic request arriving at the server during the experiment?

Solution:

Since inter-arrival times are exponentially distributed and the service times follow a normal distribution, we can model the system using an M/G/1 model.

We first compute $A = \frac{1}{2}\left[1 + \left(\frac{\sigma_{T_s}}{T_s}\right)^2\right] = \frac{1}{2}\left[1 + \left(\frac{1.66}{6.66}\right)^2\right] = 0.53$

Next, we compute $q = \frac{\rho^2 A}{1-\rho} + \rho = 1.39$

Finally, we compute $T_q = \frac{q}{\lambda} = 13.9$ ms.

A1B2C3D4E5F6G7H8I9J0

Problem 3

As the midterm date approaches, the CS350 online textbook begins to see more traffic, which could pose a problem to the stability of the server it runs on. Suppose incoming requests follow a Poisson distribution, have exponentially distributed service time, and are immediately placed in a queue to be serviced. The server will then process a request by loading the content of the textbook. When loading is done, the request is complete and can leave the system. On average, it takes 120 ms to load the textbook. We want to ensure that the textbook server won't crash so close to the midterm!

(a) **[5 points]** Consider that, on average, students place requests at a rate of 8 requests per second. What is the average amount of time that a generic request will spend in the system?

Solution:
Given assumptions tell us that this is an M/M/1 System.

In order to get T_q, we need to find the utilization of the server.

We know that $T_s = 0.12$ and $\lambda = 8$. Thus:

$$\rho = \lambda \cdot T_s = 8 \cdot 0.12 = 0.96$$
$$q = \frac{0.96}{1 - 0.96} = 24 \text{ since } q = \frac{\rho}{1-\rho}$$
$$T_q = \frac{q}{\lambda} = \frac{24}{8} = 3$$

(b) **[5 points]** The staff decides that this is too long a wait time for each student, and choose to modify the server to limit the queue to 16 requests in the system at a time. Compute the speedup of the new system (considering only properly served requests i.e., requests that don't get rejected) compared to the old system.

Solution:
With a limited queue size, this system shifts from an M/M/1 model to an M/M/1/K, with $K = 16$. In order to compute the speedup of accepted requests, we first need to compute the average number of requests in the system, and the arrival rate of accepted requests:

A1B2C3D4E5F6G7H8I9J0

$$\begin{aligned} q &= \frac{\rho}{1-\rho} - \frac{(K+1)\rho^{K+1}}{1-\rho^{K+1}} \\ &= \frac{0.96}{1-0.96} - \frac{(16+1)0.96^{16+1}}{1-0.96^{16+1}} \\ &\approx 7.03 \end{aligned}$$

$$\Pr[S_k] = \frac{(1-0.96)\cdot 0.96^{16}}{1-0.96^{17}} \approx 0.04$$

$$\begin{aligned} \lambda' &= \lambda \cdot (1 - \Pr(S_k)) \\ &= 8 \cdot (1 - \frac{(1-0.96)\cdot 0.96^{16}}{1-0.96^{17}}) \\ &\approx 7.67 \end{aligned}$$

We can now use the new λ' to compute the new T_q:

$$T_q = \frac{q}{\lambda'} = \frac{17}{7.67} \approx 0.92$$

Lastly, we can use the response time from **(a)** to compute the speedup of the new system:

$$\frac{T_{old}}{T_{new}} = \frac{3}{0.92} = 3.26\times$$

(c) **[6 points]** Over the span of an hour, what is the expected *amount of time* during which the NEW server is busy serving a request?

Solution:

The easy way to answer this question is by recalling that only accepted requests contribute to the server's utilization.

From the previous part, we know that $\lambda' = 7.67$ req./sec.

Thus the effective server utilization is $\rho' = \lambda' \cdot T_s = 7.67 \cdot 0.12 \approx 0.92$.

As such, in a 1-hour period, the server will be busy for 0.92 hours, a.k.a. 55.2 minutes, a.k.a. 3312 seconds.

Alternative solution: to say that a server is busy is equivalent to the complement of the event that the server is empty. Hence, we can find the probability that the server is busy by finding $1 - \Pr[S_0]$ and applying that probability to our chosen timespan.

Copyright ©2024 by Renato Mancuso. All rights reserved.

A1B2C3D4E5F6G7H8I9J0

$$\begin{aligned} 1 - \Pr[S_0] &= 1 - \frac{(1-\rho)\cdot\rho^0}{1-\rho^{16+1}} \\ &= 1 - \frac{0.04}{1-0.96^{17}} \\ &\approx 1 - 0.08 \\ &= 0.92 \end{aligned}$$

We find once again that the system is expected to be busy about 92% of the time, and over an hour this means it is expected to be busy for $0.92 \cdot 60 = 55.2$ minutes.

(d) **[6 points]** True/False: If the average service time of the system doubles, will the number of requests that get rejected also be exactly double compared to what it was before? Motivate your answer.

Solution:
If T_s doubles, then $\rho = 1.92$ for our system also doubles. Let us compute the probability of rejection for the new system, and compare it to the $\Pr[S_k]$ we found in **(b)**:

$$\Pr S_k = \frac{(1-1.92)\cdot 1.92^{16}}{1-1.92^{17}} \approx 0.48$$

FALSE, because $\Pr[S_{kNew}] > 2 \cdot \Pr[S_{kOld}]$

A1B2C3D4E5F6G7H8I9J0

Problem 4

You are a member of an elite computer engineering team behind a growing social networking platform PetBuddy. You manage the system which uploads and processes images of users pets'. Your current system allows users to upload pet pictures of a fixed size (exactly 250×250 pixels). The system that processes these profile pictures is very predictable, and you find that uploading one profile picture always takes 0.1 second. On average, the PetBuddy system receives 5 requests every second to upload a new pet picture.

(a) **[6 points]** What is the average response time a user will see when uploading a picture of their pet?

Solution:
We can represent PetBuddy system as an M/D/1. $\lambda = 5$ $T_s = 0.1$ $\rho = \lambda T_s = .5$ $q = \frac{\rho^2}{2(1-\rho)} + \rho = .75$ $T_q = q/\lambda$ $T_q = .75/5 = 0.15$ seconds

A1B2C3D4E5F6G7H8I9J0

(b) [8 points] Your team is considering an upgrade to the PetBuddy system (called PetBuddy-Unlimited) that allows users to upload pet pictures of any size. To do this, they have purchased an additional image-processing server that is identical to the old one. In total, the PetBuddy-Unlimited system will have two servers which can process requests in parallel from a single shared queue. Because images sizes are not fixed anymore, your team has observed that it takes each server 0.15 seconds to service each upload request, and that these service times are now exponentially distributed.

If each server in the PetBuddy-Unlimited system, when being actively utilized, consumes 20 Watts of power, what is the probability that at a random point in time the system will be consuming 40 Watts of power?

Solution:

We can represent the PetBuddy-Unlimited system as an M/M/N model, with N = 2 and $T_s = 0.15$.

To consume 40 Watts of power, both servers must be in use.

To compute the probability that both servers are busy, we can compute the parameter C. Thus, we compute:

$\lambda = 5$

$T_s = 0.15$

$\rho = \lambda T_s / N = .375$

$K = \frac{1+2\rho}{1+2\rho+\frac{(2\rho)^2}{2}} = \frac{1.75}{2.0313} \approx 0.86$

$C = \frac{1-K}{1-\rho K} = \frac{1-0.86}{1-0.375 \cdot 0.86} \approx 0.207$

(c) [8 points] If your team expects the arrival rate of requests to remain the same as in the old system, what is the speedup of PetBuddy-Unlimited in comparison to the old PetBuddy, as perceived its the users?

Solution:
We again recall that we can represent the PetBuddy-Unlimited system as an M/M/N.

We have computed the quantity $C = 0.207$ in the previous part. Next, we compute:

$q = C\frac{\rho}{1-\rho} + N\rho = 0.207\frac{0.375}{1-0.375} + 2 \cdot 0.375 \approx 0.87$

$T_q = q/\lambda$

$T_q = .87/5 = 0.174$ seconds

$slowdown = \frac{T_{new}}{T_{old}} = .174/.15 = 1.16$

Users arriving at PetBuddy Unlimited see a 1.16× slowdown.

A1B2C3D4E5F6G7H8I9J0

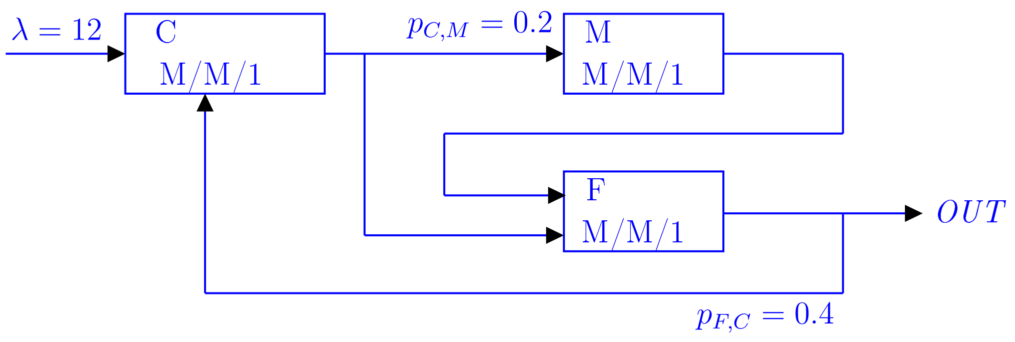

Figure 3: ChatLPT system diagram.

Problem 5

The ChatLPT is a web-based chatbot providing life pro tips. The service is quite popular, receiving on average 12 prompts every second! When a new prompt is submitted to ChatLPT, it *generally* undergoes three processing stages. First, the input is looked up in a cache at the cache machine (C), which takes about 40 milliseconds. After C, two things can happen: (i) with probability 20%, a new LPT needs to be generated, (ii) otherwise LPT generation can be skipped and some previously generated LPT is taken from the cache and forwarded to the last stage (F). In case (i), the main large language model (M) is queried for a candidate reply, which takes on average 230 milliseconds. After (M) or in case (ii) above, the model output is analyzed by a content filter (F) model which ensures that no inappropriate content is returned to the user. Querying the F model takes about 49 milliseconds. If content moderation is successful, the response can be returned to the user and the original request leaves ChatLPT. Conversely, if inappropriate content is detected, the original query is internally modified and resubmitted to the first stage (C) as if it was a new request. This is necessary because it seems that the main model produces inappropriate responses about 40% of the time. All sub-systems are implemented using single-CPU machines.

(a) **[5 points]** Provide a diagram of the system which shows the inter-connections between the C, M, and F machines. Describe the queuing models used for each machine, the known parameters, and the aossumptions you will employ to solve the system.

Solution:

Assumptions:

(1) All queues are of infinite size;

(2) Arrivals from the outside are Poisson;

(3) All service times are exponentially distributed;

(4) Requests are scheduled using the FIFO discipline;

(5) System is able to reach steady-state (to be verified).

A1B2C3D4E5F6G7H8I9J0

(b) [5 points] Identify the resource that constitutes the system bottleneck. Show your reasoning.

Solution:
In order to find the bottleneck, we first need to compute the rate of requests hitting the various resources when considering re-circulation within the system.

The input rate of new requests is $\lambda = 12$ req./sec.

A request that arrives at the last stage might be sent back to the first stage with probability $p_{F,C} = 0.4$.

We also notice that anything that goes through machine C also goes through machine F. Thus, $\lambda_C = \lambda_F$.

We can set up: $\lambda_C = \lambda + p_{F,C}\lambda_C \rightarrow \lambda_C = \frac{\lambda}{1-p_{F,C}} = \frac{12}{1-0.4} = 20$ req./sec.

Thus, $\lambda_F = \lambda_C = 20$ req./sec.

And finally $\lambda_M = p_{C,M}\lambda_C = 0.2 \cdot 20 = 4$ req./sec.

The various resource utilizations are:

(1) $\rho_C = \lambda_C \cdot T_C = 0.8$

(2) $\rho_M = \lambda_M \cdot T_M = 0.92$

(3) $\rho_F = \lambda_F \cdot T_F = 0.98$

Thus, the resource that constitutes the bottleneck is the filter machine (F).

(c) [5 points] What is the response time that should be expected by a generic user while the system is at steady state?

Solution:
We want to compute T_q that is easy to compute using Little's Law applied to the entire system.

First we use the utilizations computed above to find the expected number of requests at each sub-system.

(1) $q_C = \frac{\rho_C}{1-\rho_C} = \frac{0.8}{1-0.8} = 4$ req.

(2) $q_M = \frac{\rho_M}{1-\rho_M} = \frac{0.92}{1-0.92} = 11.5$ req.

(3) $q_F = \frac{\rho_F}{1-\rho_F} = \frac{0.98}{1-0.98} = 49$ req.

Next, we compute $q_{TOT} = q_C + q_M + q_F = 64.5$ req.

Finally $T_{q,TOT} = \frac{q_{TOT}}{\lambda} = 5.375$ sec.

A1B2C3D4E5F6G7H8I9J0

(d) **[6 points]** Consider a user that has been able to have their query processed without suffering any queuing delay. How much faster will the system appear to this user compared to the average user using the system at steady state?

Solution:

The easiest way to answer this question is to first compute the response time without queuing, call it $T_{s,TOT}$. Then, we can compute the speedup from the lack of queuing $speedup = \frac{T_{q,TOT}}{T_{s,TOT}}$.

First, compute $w_{TOT} = w_C + w_M + w_F = (q_C - \rho_C) + (q_M - \rho_M) + (q_F - \rho_F) = (4 - 0.8) + (11.5 - 0.92) + (49 - 0.98) = 61.8$

Next, $T_{w,TOT} = \frac{w_{TOT}}{\lambda} = 5.15$ sec.

Thus, $T_{s,TOT} = T_{q,TOT} - T_{w,TOT} = 0.225$ sec.

Finally $speedup = \frac{T_{q,TOT}}{T_{s,TOT}} = \frac{5.375}{0.225} \approx 23.89\times$ faster.

NOTE: in this particular case, an alternative way to compute the same value of $T_{s,TOT}$ is by considering the expected number of times that a request loops back into the system $k = \frac{1}{1-p_{F,C}}$. This is the mean of a random variable which counts the number trials (including the last successful trial) and that follows a geometric distribution with probability of failure $p_{C,M}$.

Hence, $T_{s,TOT} = k(T_C + p_{C,M}T_M + T_F) = \frac{0.04+0.2\cdot0.23+0.049}{1-0.4} = 0.225$ sec.

(e) **[4 points]** What is maximum throughput that can be sustained by the system without losing its ability to reach steady state?

Solution:

The maximum throughput (a.k.a. the capacity) of the system is determined by its bottleneck. We have already identified that the bottleneck is the F machine.

To find the maximum throughput it is enough to find the maximum request rate of new requests λ^* such that $\rho_F = \frac{T_F \lambda^*}{1-p_{F,C}} = 1$.

It follows that $\lambda^* = \frac{1-p_{F,C}}{T_F} = \frac{0.6}{0.049} \approx 12.245$ req./sec.

A1B2C3D4E5F6G7H8I9J0

[EXTRA BLANK PAGE]

Copyright ©2024 by Renato Mancuso. All rights reserved.

A1B2C3D4E5F6G7H8I9J0

Some Probability Density Functions

- Poisson: $f(x) = \frac{\lambda^x}{x!}e^{-\lambda}$
- Exponential: $f(x) = \lambda e^{-\lambda x}$, $F(x) = 1 - e^{-\lambda x}$
- Standard Normal: $f(z) = \frac{1}{\sqrt{2\pi}}e^{-0.5z^2}$

Equations for Some Queuing Systems

- M/G/1 System: $q = \frac{\rho^2 A}{1-\rho} + \rho$, $w = \frac{\rho^2 A}{1-\rho}$, where $A = \frac{1}{2}\left[1 + \left(\frac{\sigma_{T_s}}{T_s}\right)^2\right]$
- M/D/1 System: $q = \frac{\rho^2}{2(1-\rho)} + \rho$, $w = \frac{\rho^2}{2(1-\rho)}$
- M/M/1/K system: $q = \begin{cases} \frac{\rho}{1-\rho} - \frac{(K+1)\rho^{K+1}}{1-\rho^{K+1}} & \text{for } \rho \neq 1 \\ \frac{K}{2} & \text{for } \rho = 1 \end{cases}$,

  $P(\text{“rejection”}) = P(S_K) = \begin{cases} \frac{(1-\rho)\rho^K}{1-\rho^{K+1}} & \text{for } \rho \neq 1 \\ \frac{1}{K+1} & \text{for } \rho = 1 \end{cases}$
- M/M/N System: $q = C\frac{\rho}{1-\rho} + N\rho$, $w = C\frac{\rho}{1-\rho}$,

  where $\rho = \frac{\lambda T_s}{N} = \frac{\lambda}{N\mu}$, $C = \frac{1-K}{1-\rho K}$, and $K = \frac{\sum_{i=0}^{N-1}\frac{(N\rho)^i}{i!}}{\sum_{i=0}^{N}\frac{(N\rho)^i}{i!}}$.