Department of Computer Science BOSTON UNIVERSITY

CS-350 - Fundamentals of Computing Systems

Midterm Exam #1
Fall 2023

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:	/20
Problem #4:	/23
Problem #5:	/23
Total Score:	/105

Remarks:

- This is a closed-book/notes exam.
- Basic calculators are allowed.
- You have 80 minutes to complete your exam.
- There are 105 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.**

Problem 1

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

	Statement	T/F
a.	In a system with a multi-processing level (MPL) of 10 and a single CPU, there can at most 1 process in the *ready* state, while every other process must be in the *blocked* state from the CPU's perspective.	**FALSE**
b.	In an M/M/1/K system, if λ is much larger than $\mu = \frac{1}{T_s}$ there is a small but non-zero probability that a generic request will not be rejected.	**TRUE**
c.	Unlike in an M/M/1 system, in an M/D/1 system there is a linear relationship between utilization and average response time at steady-state.	**FALSE**
d.	The bottleneck of a complex system can be identified as the resource with the highest utilization when the system is operating at steady-state.	**TRUE**
e.	The capacity of a complex system can change if the characteristics of the workload submitted by its users change, even if the system hardware remains unchanged.	**TRUE**
f.	The utilization of an M/G/1 system at steady state can be halved by adding a firewall in front of the system that discards 50% of the incoming requests.	**TRUE**
g.	Call X a Poisson-distributed variable with mean α. Then X can be interpreted as the number of events that occur in a given time unit; and the time in-between events is also a Poisson-distributed random variable with mean $1/\alpha$.	**FALSE**
h.	The availability of a system is defined as the probability that the system will stay operational for a continued lapse of time.	**FALSE**
i.	The throughput at steady state can only asymptotically approach the system capacity but never perfectly match it.	**TRUE**
j.	For intermediate values of utilization, an M/M/N system will provide better quality of service compared to an N*M/M/1 system when they are both subject to the same traffic and service conditions.	**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.

Problem 2

Take a look at the 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 process. 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.

```
1  struct request {
2    uint64_t req_id; /* Integer request ID set by the client */
3    struct timespec req_timestamp; /* time when request sent by client  */
4    struct timespec req_length; /* time length of the request */
5  };
6
7  struct request_meta {
8    struct request request;
9    struct timespec receipt_timestamp; /* time when request enqueued */
10   struct timespec start_timestamp;   /* time when request starts service */
11   struct timespec completion_timestamp; /* time when request completed */
12 };
13
14 struct worker_params {
15   int conn_socket;
16   struct queue * the_queue;
17 };
18
19 /* Main logic of the worker thread */
20 int worker_main (void * arg)
21 {
22   struct timespec now;
23   struct worker_params * params = (struct worker_params *)arg;
24
25   while (/* Worker termination condition */) {
26     struct request_meta req;
27     struct response resp;
28
29     /* 1 */ -----------------------------------------------------
30
31     /* 2 */ -----------------------------------------------------
32     busywait_timespec(req.request.req_length);
33
34     /* 3 */ -----------------------------------------------------
35
36     resp.req_id = req.request.req_id;
37
38     /* 4 */ -----------------------------------------------------
39     printout_completed_request(req);
40     dump_queue_status(params->the_queue);
41   }
42   return EXIT_SUCCESS;
43 }
44
45 void handle_connection(int conn_socket)
46 {
47   struct queue * the_queue;
48   int in_bytes, worker_id;
49   struct request_meta * req = (struct request_meta *)malloc(sizeof(struct request_meta));
50
51   the_queue = initialize_queue(QUEUE_SIZE);
52   worker_id = start_worker(conn_socket, the_queue);
53
54   do {
55     in_bytes = recv(conn_socket, &req->request, sizeof(struct request), 0);
56
57     /* 5 */ -----------------------------------------------------
58
59     if (in_bytes > 0) {
60
61       /* 6 */ -----------------------------------------------------
62     }
63   } while (in_bytes > 0);
64 }
```

Listing 1: Handle connection and worker implementation

(a) [12 points] Use the list of lines in the last page to complete the blanks in the code. You can use the line letter (A, B, ...) instead of writing the full line in the code above.

Solution:

Blank 1: Line B.

Blank 2: Line E.

Blank 3: Line G.

Blank 4: Line L.

Blank 5: Line J.

Blank 6: Line D.

(b) [4 points] Take a look at the printout produced by the server for these 5 requests. The format is the following, where (1) `<sent ts>`, (2) `<receipt ts>`, (3) `<start ts>`, and (4) `<completion ts>` are the timestamps at which the request was (1) sent by the client, (2) recevied by the parent thread, (3) started being worked on, and (4) completed by the worker thread.

```
R<request ID>:<sent ts>,<req. length>,<receipt ts>,<start ts>,<completion ts>
```

Server printout:

```
R0:5670986.970837,5.000000,5670986.971040,5670986.971085,5670991.971086
Q:[R1]
R1:5670989.970890,2.000000,5670989.971019,5670991.971141,5670993.971145
Q:[]
R2:5670996.970949,3.000000,5670996.971074,5670996.971147,5670999.971152
Q:[]
R3:5671001.970993,5.000000,5671001.971101,5671001.971173,5671006.971174
Q:[R4]
R4:5671003.971029,1.000000,5671003.971142,5671006.971232,5671007.971233
Q:[]
```

What is the estimated utilization of the server?

Solution:

For the total busy time, we can sum up all the request lengths: $5 + 2 + 3 + 5 + 1 = 16$

For the total time window of reference, we can use the difference between the timestamp of completion of the last request and the timestamp at which the first request was sent. That is: $5671007.971233 - 5670986.970837 = 21.000396 \approx 21$.

The utilization is therefore: Util. $= 16/21 = 0.7619 \approx 0.762$.

(c) [4 points] Looking at the same printout provided above, what is the average response time seen by the requests sent by the client?

Solution:
To compute the average response time of the $N = 5$ requests, we can compute the individual response times and average them out.

R0: $5670991.971086 - 5670986.970837 \approx 5$; R1: $5670993.971145 - 5670989.970890 \approx 4$;
R2: $5670999.971152 - 5670996.970949 \approx 3$; R3: $5671006.971174 - 5671001.970993 \approx 5$;
R4: $5671007.971233 - 5671003.971029 \approx 4$;

So the average response time is $\frac{5+4+3+5+4}{5} = 4.2$.

(d) [5 points] After oberving the system for a while, you have discovered that the distribution of inter-arrival times results in the plot provided in Figure 1. Apart from the distribution of the experimental data, some theoretical distributions have been added to the plot for your convenience. Moreover, with an experimentally measured server utilization of 91.895%, the average response time appears to be 0.5 sec.

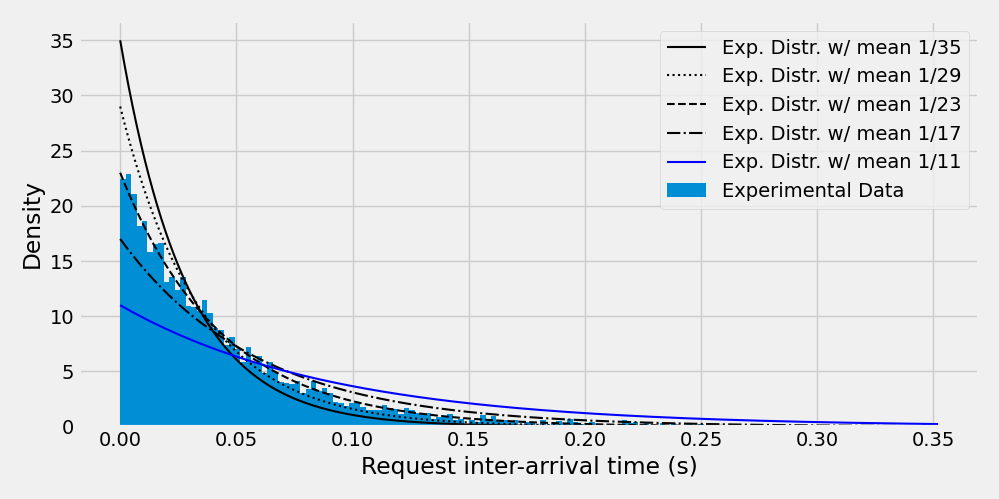

Figure 1: Experimental and theoretical inter-arrival time densities.

Would an M/M/1 model be a good approximation for the steady-sate behavior of this system? Motivate your answer, and if so, provide its parameters (1) λ and (2) T_s.

Solution:
From the plot, we can see that the experimental data fits quite well the shape of a theoretical exponential distribution with mean 1/23, thus $\lambda = 23$.

To verify that the model matches an M/M/1 system, we can check that $T_q = \frac{1}{\lambda} \cdot \frac{\rho}{1-\rho}$, with ρ given as $\rho = 0.91895$ and $T_q = 0.5$ sec.

$\frac{1}{\lambda} \cdot \frac{\rho}{1-\rho} = 11.338062924/23 = 0.4929$

Thus, the experimental value of T_q is close enough to the theoretical value for the system to be modeled well by an M/M/1 model.

Thus, $T_s = \frac{\rho}{\lambda} = 0.03995 \approx 0.04$.

Problem 3

A complex transactional system accepts requests from its users and employs three servers in its backend to take care of requests. When a request arrives, it is routed on of the three backend servers to be processed. After benchmarking the system for a while, you have noticed that 30 out of 100 requsts are routed to Server A, 20 out of 100 to Server B, and 50 out of 100 to Server C. When a request is routed to Server A, the response time is 10 ms; it is 5 ms when routed to Server B, and 20 ms if routed to Server C. Answer the following.

(a) [5 points] What is the probability that a request will have a response time greater than 6 ms?

Solution:
To have a response time t_q greater than 6 ms, the request needs to be routed through either Server A or C. Thus:

$P(t_q > 6) = p_A + p_C = 30/100 + 50/100 = 0.3 + 0.5 = 0.8$

(b) [7 points] Call N the number of consecutive requests that will need to be sent, on average, before the $(N+1)^{th}$ request sees a response time of 5 ms. What is a good estimate for N?

Solution:
N is a geometrically distributed variable. The probability of failure p_{fail} is the probability that a request will see a reponse time different from 5 ms, i.e., $p_{fail} = p_A + p_C$.

The expected value of N is $E[N] = \frac{p_{fail}}{1-p_{fail}} = 4$.

(c) [8 points] Server A and C are unmodified, but Server B has been upgraded to have a mean response time of 3 ms. But instead of being a constant, the response time is now distributed following an exponential distribution with mean 3 ms. What is the probability that a generic request arriving at the entire system (not just Server B!) will experience a response time of 2 ms or less.

Solution:
For this, we first figure out the probability of seeing a response time of 2 ms or less when routed to Server B, call it t_q^B.

We use the CDF of an exponential distribution with parameter $\lambda = 1/mean = 1/3$. We compute: $P(t_q^B \leq 2) = F(2) = 1 - e^{-2/3} = 0.487$.

The probability that the response time t_q for a generic request at the whole system is less than or equal to 2 ms is:

$P(t_q \leq 2) = p_B \cdot P(t_q^B \leq 2) = 0.2 \cdot 0.487 = 0.0974$.

Problem 4

A single-processor system has a single queue of requests and you have determined that at steady state the average total number of requests in the system (queued or in service) is 19 and that the average service time is 50 s.

(a) [4 points] If you wanted to employ an M/M/1 model to study this system, what assumptions do you need to make?

Solution:

1. Infinite queue size;
2. Poisson arrivals a.k.a. exponentially distributed inter-arrival times;
3. Exponentially distributed service times;
4. FIFO queue dispatch policy;

(b) [7 points] From now on, assume that such asusmptions hold. What is the throughput of the system at steady state? **Solution:**

When the system is at steady state, the throughput matches the arrival rate λ.

Since we know that $q = 19 = \frac{\rho}{1-\rho}$, we can write the relationship $\rho = \frac{q}{q+1} = \frac{19}{20} = 0.95$.

We know that $T_s = 50\ s$ and that $\rho = \lambda T_s$, thus $\lambda = \frac{\rho}{T_s} = 0.019$ req. per seconds.

(c) [6 points] What is the slowdown that users experience due to queuing effects compared to the amount of service they request?

Solution:

We can use Little's Law to compute the steady-state average response time.

$T_q = \frac{q}{\lambda} = 19/0.019 = 1000$ seconds.

The slowdown due to queuing can be computed as $slowdown = \frac{T_q}{T_s} = 20\times$

(d) [6 points] What is the probability that, when a request is sent, it starts processing right away and that it will take 60 seconds or more to complete processing.

Solution:

The probability of starting processing right away is the probability of arriving at an empty queue, which is $1 - \rho = 1 - 0.95 = 0.05$.

To compute the probability that, once a request starts processing, the request will take 60 s or more to complete, we use the CDF of an exponential distribution with parameter $\mu = 1/T_s$.

$P(t_s > 60s) = 1 - P(t_s \leq 60s) = 1 - F(60) = 1 - (1 - e^{-\frac{60}{50}}) = 1 - 0.699 = 0.301$

Thus the probability of starting processing right away and processing for 60 s or more is: $0.05 \cdot 0.301 = 0.01505$.

Problem 5

A government service has been initially deployed an a single-queue, single-server system. Call the server S0. The service receives on average, at steady state, about 8280 requests per hour. And it turns out that the number of requests issued by the citizens using the service is Poisson-distributed. You have benchmarked that the service time at steady state is exponentially distributed with an average of 0.4 seconds. Your job is to cost-efficiently optimize the system.

(a) **[6 points]** First off, compute the average queue length that the typical user will experience at steady state upon submitting a request?

Solution:

First, we compute $\lambda = 8280/(60 * 60) = 2.3$ req./second.

Next, we recognize that $T_s = 0.4$ seconds.

For this part, we first compute $\rho = \lambda \cdot T_s = 0.92$

Finally, we can compute $q = \frac{\rho}{1-\rho} = \frac{0.92}{1-0.92} = 11.5$.

NOTE: Solutions that computed w instead of q are also valid!

(b) **[6 points]** In an attempt to optimize the performance of the system, you propose the purchase of one more server, call it S1, with identical specifications as S0. The idea is to have all the user requests enqueued in a single, central queue. The request at the head of the queue is then served by S0 or S1, as soon as one of them becomes available. Because the two servers are identical, each will still process a generic request in 0.4 seconds, on average. What queue length will a generic user experience upon submitting a request?

Solution:

In this case, we are dealing with an M/M/2 system.

In this case, the utilization of each server is $\rho' = \frac{\lambda T_s}{2} = 0.46$.

We can compute $K = \frac{1+2\rho'}{1+2\rho'+\frac{(2\rho')^2}{2}} = \frac{1.92}{2.3432} = 0.819$

Next, we compute $C = \frac{1-K}{1-\rho' K} = \frac{0.181}{0.623} = 0.29$

And finally $q' = C\frac{\rho'}{1-\rho'} + N\rho' = 0.29 \cdot 0.85 + 0.92 = 1.167$

NOTE: Solutions that computed $w' = q' - N\rho'$ instead of q' are also valid!

(c) **[6 points]** An alternative is to decommission the already-deployed server S0 and to purchase 4 new servers. The 4 new servers (all together!) cost half than S1 because they are much slower. Indeed, each will process a request in 0.9 seconds, on average. In this case, you propose that requests are randomly pre-split to the 4 servers, thus each

server now has a per-server queue. What queue length will a generic user experience upon submitting a request?

Solution:
The new organization can be modeled using a 4*M/M/1 system model.

The new $T_s' = 0.9$ sec.

In this case, we recompute the per-server utilization as: $\rho'' = \frac{\lambda T_s'}{4} = 0.5175$

A generic user will be enqueued in one of these M/M/1 systems, thus the queue length they will experience is: $q'' = \frac{\rho''}{1-\rho''} = 1.073$.

NOTE: Solutions that computed w'' instead of q'' are also valid!

(d) **[5 points]** Which solution between what you have considered in Part **(b)** and Part **(c)** is more desirable? Motivate your answer.

Solution:
The solution proposed in Part **(c)** is more desirable because it's both the cheaper option and provides better quality of service to the citizens, if the length of the queue as perceived by a new customer is the metric chosen.

On the other hand, if we consider the averege response time as the chosen metric to reason abot the system's quality of service we have:

1. For Part **(b)**: $T_q' = q'/\lambda = 1.167/2.3 = 0.5$ sec.
2. For Part **(c)**: $T_q'' = \frac{q''\cdot 4}{\lambda} = 1.866$ sec.

If we look at the response time, the 4 servers are half the cost of S1, but the response time seen by the users would be 3.7× worse compared to the solution proposed in Part **(b)**. Thus, one could conclude that the solution in part Part **(b)** is a better cost/performance trade-off.

NOTE: Solutions that reasoned on w' vs. w'' and/or T_w' vs. T_w'' are also valid. The conclusions are the same.

[EXTRA BLANK PAGE]

[COMPLETION LINES FOR PROBLEM 1]

You can detatch this page for your convenience.

A. `req = get_from_queue(params.the_queue);`

B. `req = get_from_queue(params->the_queue);`

C. `req = get_from_queue(the_queue);`

D. `add_to_queue(*req, the_queue);`

E. `clock_gettime(CLOCK_MONOTONIC, &req.start_timestamp);`

F. `clock_gettime(CLOCK_MONOTONIC, &req->start_timestamp);`

G. `clock_gettime(CLOCK_MONOTONIC, &req.completion_timestamp);`

H. `clock_gettime(CLOCK_MONOTONIC, &req->completion_timestamp);`

I. `clock_gettime(CLOCK_MONOTONIC, &req.receipt_timestamp);`

J. `clock_gettime(CLOCK_MONOTONIC, &req->receipt_timestamp);`

K. `send(params->conn_socket, &req.request, sizeof(struct request), 0);`

L. `send(params->conn_socket, &resp, sizeof(struct response), 0);`

M. `recv(params->conn_socket, &req.request, sizeof(struct request), 0);`

N. `recv(params->conn_socket, &resp, sizeof(struct response), 0);`

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 $K = \frac{\sum_{i=0}^{N-1} \frac{(N\rho)^i}{i!}}{\sum_{i=0}^{N} \frac{(N\rho)^i}{i!}}$.

