IE306 Systems Simulation

Homework 2

29.10.2022

Tevfik Buğra Türker

2019402120

Question 1

A_1 0,380 D_2 0,391 A_4 0,975 0,593

D_1 0,496 A_3 0,020 D_4 0,759 0,560

A_2 0,832 D_3 0,480 0,905

* If both servers are idle, prefer Server 1.
* Initially, the first server is busy.

Interarrival ~ U(6,16)

Server 1 ~ U(14,21)

Server 2

0,18	8
0,48	12
0,78	22
1	33

State variables

Time	LQ	LS_1	LS_2	Int Arr	Ser1	Ser2	Future event list A	D_1	D_2	Set	Time in Q
0	0	1	0	9,8	17,472	-	9.8	17,472	∞	0	-
9,8	0	1	1	14,32	-	12	24,12	17,472	21,8	0, 9,8	-
17,472	0	0	1	-	-	-	24,12	∞	21,8	9,8	-
21,8	0	0	0	-	-	-	24,12	∞	∞		-
24,12	0	1	0	6,2	17,36	-	30,32	41,48	∞	24,12	-
30,32	0	1	1	15,75	-	22	46,07	41,48	52,32	24,12, 30,32	-
35	0	1	1	-	-	-	46,07	41,48	52,32	24,12, 30,32	-

Question 2

Before diving into question 2a, I would like to briefly mention how the code I wrote works. In the code, there are two main functions: arrival and departure. These functions arrange what will happen if one of these events occur. Their basic working principle is summarized in the table below. Besides, since in our case there are two servers, the functions that I defined can represent these two servers.

Arrival and Departure

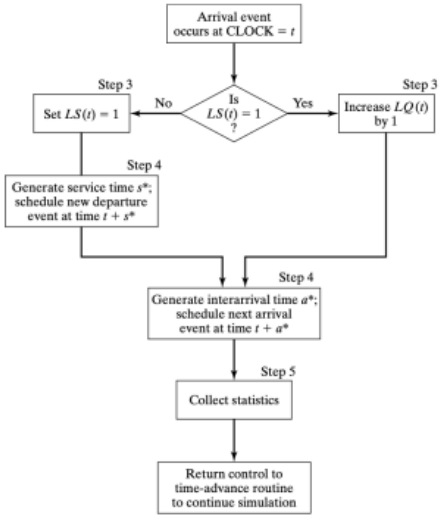

Figure 5 Execution of the arrival event.

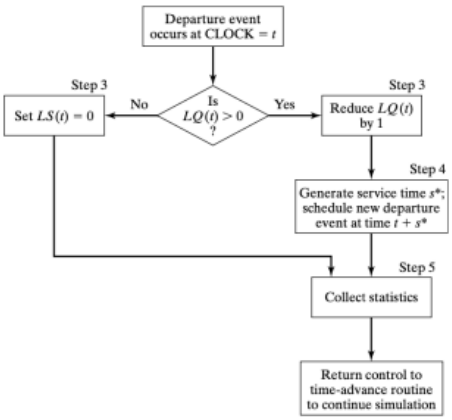

Figure 6 Execution of the departure event.

Apart from these functions, in the code there is a "while loop" which decides whether there should be an arrival or a departure event. This decision is made by checking which future event is the closest to the current time. This iteration continues until n^{th} customer arrives.

```
while n<999:
    if future_event["A"] < future_event["D1"] and future_event["A"] < future_event["D2"]:
        arrival(future_event_id["A"])
        n +=1
    elif future_event["D1"] < future_event["A"] and future_event["D1"] <
future_event["D2"]:
        departure(future_event_id["D1"])
    elif future_event["D2"] < future_event["A"] and future_event["D2"] <
future_event["D1"]:
        departure(future_event_id["D2"])
```

Eventually, all the information related to the customer n is overwritten on an initially empty dictionary. This dictionary has the key value n and every n value has a list which contains specific details about that customer.

```
custdict = {}
for i in range(0,1000):
    custdict[i]=[0,0,0,0,0,0,0,0]

# {customer number:[0]=interarrival time, [1]=arrival time, [2]=service beginning time,
[3]=service time, [4]=departure time, [5]=server number, [6]=time in the system, [7]= time
in the queue]}
```

Also, to keep track of events, I defined 4 different dictionaries. These are:

1) the "*current*" dictionary which keeps the number of customers
2) the "*future_event*" dictionary which keeps the closest future event times
3) "*future_event_id*" dictionary which keeps the order of the customers that are assigned in the previous dictionary
4) "*queue_dict*" which keeps the order of the queue.

```
current = {"Ls1":0, "Ls2":0, "Lq":0} # {"Ls1": number of customers in server 1, "Ls2":
number of customers in server 2, "Lq": number of customers queue
future_event = {"A":0, "D1":999, "D2":999} # {"A": the nearest arrival, "D1": the nearest
departure from server 1, "D2": the nearest departure from server 2}
future_event_id = {"A":0, "D1":0, "D2":0} # {keys of the nearest events}
queue_dict = {} # order and the arrival times of the customers that are in the queue
```

For the first part of the 2nd question, in order to use given random numbers I added

```
rand_numb_list=[0.380,0.496,0.832,0.391,0.020,0.480,0.975,0.759,0.905,0.593,0.560]
```

and when I used one, I removed that number from the list and used the next one.

```
rand_numb_list.remove(rand_numb_list[0])
```

Note: I added the full code in the appendix file.

Question 2a

I run my simulation for 35 minutes with inputting and using the random numbers given in Question 1.

My code output:

0: [0, 0, 0, 17.472, 17.472, 1, 17.472, 0]

1: [9.8, 9.8, 9.8, 12, 21.8, 2, 12.0, 0.0]

2: [14.32, 24.12, 24.12, 17.36, 41.48, 1, 17.36, 0.0]

3: [6.2, 30.32, 30.32, 22, 52.32, 2, 22.0, 0.0]

4: [15.75, 46.07, 46.07, 18.151, 64.221, 1, 18.151, 0.0]

Representation of the numbers given above:

customer number: [0]=interarrival time, [1]=arrival time, [2]=service beginning time, [3]=service time, [4]=departure time, [5]=server number, [6]=time in the system, [7]= time in the queue]

Comment: When I compare my Question 1 values with my code output, I see that every value is the same for both. For this case, I can verify that the code I wrote works properly.

Question 2b

1) Average time spent in the queue

The code:

```
#Average time spent in the queue

for y in range(0,1000):
    if custdict[y][1] > 7000:
        del custdict[y]

sum_of_queue_times = 0
for m in range(0,len(custdict)):
    sum_of_queue_times += custdict[m][7]

print(sum_of_queue_times/len(custdict))
```

To calculate the average time spent in the queue, I summed up waiting times of individual customers and divided it to total customer count in 7000 minutes.

Code results:

2.9834163238777376

2.361235877706449

1.9065863197734074

2.9718717760789337

Mean: 2.555777574

My estimation for the average time spent in the queue is **2.555777574**.

2) Average number of customers in the system

The code:

```
#Average number of customers in the system

for y in range(0,1000):
    if custdict[y][1] > 7000:
        del custdict[y]

sum_of_system_times = 0
for m in range(0,len(custdict)):
    sum_of_system_times += custdict[m][6]

print(sum_of_system_times/7000)
```

To calculate the average number of customers in the system, I added up all the system times and divided it by 7000.

Code results:

1.8439091330716655

1.8089854936120704

2.0075620701620274

1.7814231491781647

Mean: 1.860469962

My estimation for the average number of customers in the system is **1.860469962**.

3) The average utilization of each server

The code:

```
#The average utilization of each server

for y in range(0,1000):
    if custdict[y][1] > 7000:
        del custdict[y]
sum_of_server1_times = 0
for m in range(0,len(custdict)):
    if custdict[m][5] == 1:
        sum_of_server1_times += custdict[m][3]
print(sum_of_server1_times/7000)
sum_of_server2_times = 0
for m in range(0,len(custdict)):
    if custdict[m][5] == 2:
        sum_of_server2_times += custdict[m][3]
print(sum_of_server2_times/7000)
```

To calculate the average utilization of each server, I added up server times for each distinct server and divided them by 7000.

Code results for server1:

0.848521958402262

0.8706980282153237

0.867667601148404

0.8404733197916132

Mean for server 1: 0.8568402269

My estimation for the average utilization of server 1 is **0.8568402269**.

Code results for server2:

0.7907142857142857

0.7548571428571429

0.7988571428571428

0.8084285714285714

Mean for server 2: 0.7882142857

My estimation for the average utilization of server 2 is **0.7882142857**.

4) Probability of a customer not waiting in the queue

The code:

```
#Probability of a customer not waiting in the queue

for y in range(0,1000):
    if custdict[y][1] > 7000:
        del custdict[y]
count_of_queue = 0
for m in range(0,len(custdict)):
    if custdict[m][7] == 0:
        count_of_queue +=1
print(count_of_queue/len(custdict))
```

To calculate the probability of a customer not waiting in the queue I summed up all the customers who did not wait in the queue and divided it by the total number of customers in 7000 minutes.

Code results:

0.6057233704292527

0.6006240249609984

0.53198127925117

0.5224111282843895

Mean: 0.5651849507

My estimation for the probability of a customer not waiting in the queue is **0.5651849507**.

Little's Law: $L = \lambda \times W$

L = 1.860469962 customer

λ = 0.09090909091 customer / min

W = 20.46516958 min

According to Little's Law, average time spent in the system (W) should be 20.46516958 min.

5) Average time spent in the system

The code:

```
#Average time spent in the system

for y in range(0,1000):
    if custdict[y][1] > 7000:
        del custdict[y]
sum_of_system_times = 0
for m in range(0,len(custdict)):
    sum_of_system_times += custdict[m][6]
print( sum_of_system_times/len(custdict))
```

To calculate the average time spent in the system, I summed up system times of individual customers and divided it to total customer count in 7000 minutes.

Code results:

19.97245195915842

20.59406201488025

20.483752460064903

21.259066950552075

Mean: 20.57733335

Comment: When the average time spent in the system is calculated with the help of Little's Law, the result is 20.46516958 minutes. This result is almost same as the value I estimated with the code which is 20.57733335. In the light of such information, I can verify that Little's Law holds for my answers above.