



UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and
Information Technology

ERP 420

RESEARCH PROJECT

PRACTICAL 2: PRIORITY QUEUEING

Name and Surname	Student Number	Signature
J.R. Gouws	16033915	

By signing this assignment I confirm that I have read and are aware of the University of Pretoria's policy on academic dishonesty and plagiarism and I declare that the work submitted in this assignment is my own as delimited by the mentioned policies. I explicitly declare that no parts of this assignment have been copied from current or previous students' work or any other sources (including the internet), whether copyrighted or not. I understand that I will be subjected to disciplinary actions should it be found that the work I submit here does not comply with the said policies.

This report will be assessed based on ECSA ELO 4.5, 5.1, 5.2 and 5.3.

ELO 4: Investigations, experiments and data analysis

1. Effective planning and execution of investigations and experiments.
2. Critical evaluation of pertinent literature.
3. Application of correct research methodology and analysis.
4. Interpretations, analyses and conclusions emanating from results and data.
5. Documentation of investigations, experiments, data, results and conclusions in a technical report.

ELO 5: Engineering methods, skills and tools, including information technology

1. Application of appropriate engineering methods.
2. Using appropriate engineering skills and tools.
3. Assessment of outcome from engineering methods, skills and tools.

Friday 20th September, 2019

1 Introduction

This practical will look at the concept of priority queuing, making use of an M/M/1 queuing simulator. Two types of priorities will be considered, known as preemptive non-resume and non-preemptive queuing. Three queuing simulators will be implemented, one for each priority scheme, as well as a simulator where no priorities are assigned to the packets. Each simulator will receive the traffic generated according to an exponential distribution with the packet size and inter-arrival times. For the priority simulators, traffic will consist of two priority levels known as high priority and low priority packets. Various arrival rates will be considered for priority levels. The simulator will determine various queuing performance parameters. The performance parameters will be used to critically analyse and compare priority simulations with each other and with the simulations using packets with no priority [1].

2 Background

In some applications of queuing systems, the system will consider all packets of being equally important. When this is the case, the servers will accept a packet according to a service discipline and finish the service before moving on the next packet. For this paper, packets having no priority will simply receive service based on the time the packet arrived. Packets received first, will be serviced first and therefore follows a first-in, first-out (FIFO) discipline. When the arrival rate is greater than the service rate of the system, a queue will form, where the simulator will service the packets in the queue according to a FIFO service discipline. When priority queuing is considered, packets with a high priority receive precedence above low priority packets. How high priority packets receive precedence will be determined by the priority scheme used, being either the preemptive non-resume priority scheme or the non-preemptive priority scheme.

2.1 Preemptive Non-resume Priority Scheduling

Following this priority scheduling strategy, low priority packets will be interrupted (preempted) when it is being processed and the system receives a high priority packet. When this is the case, the server immediately discontinues the servicing of the low priority packet, discards all progress of the low priority packet and places the low priority packet back into the front of the queue. The system will not consider processing low priority packets while there are still high priority packets in the queue. The system will only start processing the low priority packets in the queue when all high priority packets received, has been processed successfully.

2.2 Non-preemptive Priority Scheduling

Similar to the preemptive non-resume priority scheduling, the server will serve all the high priority packets that have arrived in the system before the low priority packets. The difference is when a low priority packet is being serviced and the system receives a high priority packet. In this case, the server first finishes servicing the low priority packet before the high priority packet is serviced.

2.3 M/M/1 Performance Parameter Notations

To compare the simulations that will be executed during this practical, a set of performance parameters associated with an M/M/1 queue will be calculated. Throughout the paper, the following notation will be used to describe the associated performance parameter being evaluated [2]:

λ – The arrival rate is the number of packets received per unit of time.

μ – The service rate is the number of packets serviced within a unit of time.

W_q – The average time a customer spends in the queue waiting.

L_q – Average number of customers waiting in line.

3 Design and Implementation

In this section, we will see how each simulator was designed and implemented. For each of the simulators, a packet class was defined, containing the attributes described in Table 1.

Attribute	Unit
Packet size	bits
Priority	High or Low
Inter-arrival time	μs
Absolute arrival time	μs
Transmission time	μs
L_q	μs
Starting time	μs
Completion time	μs
Response time	μs

Table 1. Description of the packet class and the associated attributes and their units.

Two trace files are generated at different average inter-arrival times. The inter-arrival times are generated using an exponential distribution with a specified mean. One trace file contains the high priority packets and the other trace file contains the low priority packets. For each of the trace files, approximately 100 seconds worth of traffic is generated. The amount of packets that are therefore generated per trace file depends on the average inter-arrival time. The number of packets to be generated is then 100 seconds divided by the average inter-arrival time chosen. Each packet in the trace file will also be allocated a size, generated with an exponential distribution with a mean value of 1000 bits. After the trace files have been generated, a simulator can be selected to process the trace files for the high and low priority packets. The design of each of the simulators is shown in Figure 1.

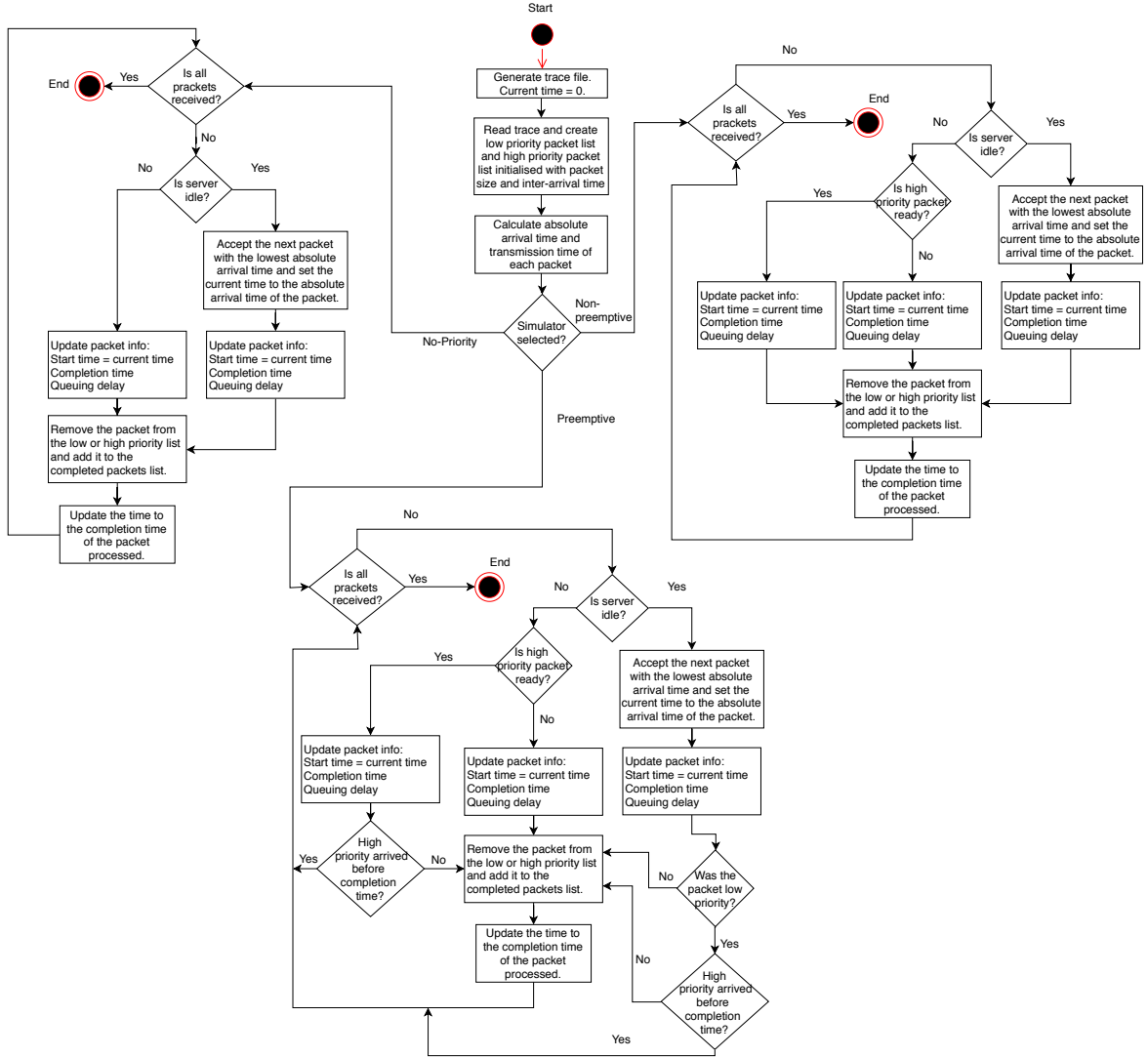


Figure 1. Flow diagram of the preemptive non-resume, non-preemptive and no priority simulators.

As seen from Figure 1, after the trace files are generated, the simulator initialises a counter to keep track of the simulator time. The trace files are then read and two lists of packets are initialised, one for high priority and one for low priority. The packets contained in the lists are initialised with the packet priority, packet size and the packet inter-arrival time. After the lists are initialised, the absolute arrival time is calculated for all packets. It is assumed that traffic for both low and high priority packets are generated starting at time zero. The absolute arrival time for each trace file is therefore separately calculated starting at time zero. Absolute arrival time is simply calculated by summing all the preceding packets inter-arrival times together, along with the considered packet inter-arrival time. For all the simulators, a link capacity of 1 Mbps was considered. The transmission times of all the packets are then calculated using:

$$\text{Transmission time} = \frac{\text{Packet size}}{\text{Link capacity}}. \quad (3.1)$$

After the preceding parameters were calculated for all the packets, we can proceed in further simulating the processing of packets by selecting the preferred simulator. The simulator will run until all the packets have been serviced successfully.

3.1 Non-preemptive Simulator

The simulator keeps track of the time that has already elapsed while processing the packets. The simulator process one packet at a time. The first thing to check when a packet is received is whether the server is busy or not. The server is busy when the new packet that is received has an absolute arrival time, less than the simulator time. The simulator first verifies if a high priority packet was received while the server is busy. If this was the case, the simulator processes the high priority packet. If not, the simulator checks whether a low priority packet was received while the server was busy. If this was the case, the low priority packet is processed. A packet is processed by the simulator by calculating the start time (current simulator time), completion time (start time and transmission time of the packet added together) and queuing delay (difference between the start time and the absolute arrival time of the packet) of the packet and then appending the packet to a list of completed packets and removing the packet from either the high or low priority queue, depending on the packet priority. The simulator time is also then updated to the completion time of the packet. When the simulator has determined to be in an idle state when the next packet is received, the simulator determines whether the next packet that is received is a low or high priority packet, by comparing the absolute arrival time of the next high and low priority packet in the respective lists. When it has been determined whether the next packet to be processed is of high or low priority, the simulator time is updated to the absolute arrival time of the packet and the packet is removed from the respective list. The packet is then processed and added to the completed packet list. The simulator time is updated again to the completion time calculated for the packet. When all the packets have been processed, the list of completed packets is used to calculate the response time of each packet, by adding the queuing delay to the transmission time of each packet.

3.2 Preemptive Non-resume Simulator

For the preemptive non-resume simulator, the method of processing the packets mostly stays the same as the non-preemptive simulator. The difference is that the low priority packets are not immediately removed from the low priority list if it is being processed. Rather than removing the packet from the list before being processed, an extra step is added after the low priority packet is processed. After processed, the simulator compares the calculated completion time of the low priority packet with the absolute arrival time of the next high priority packet. If the absolute arrival time of the next high priority packet is less than the completion time of the low priority packet, the low priority packet is not added to the completed packets lists, but rather returns to the front of the low priority packet list to be reprocessed. The system time is then set to the absolute arrival time of the high priority packet, as this case will only occur when the system is in idle or busy processing the low priority packet, which is interrupted at the absolute arrival time of the high priority packet.

3.3 No Priority Simulator

For the case where no priorities are allocated to the packets, two trace files will still be generated in the same manner that we created the trace files for priority queuing. This is to effectively compare the priority simulators to the no priority simulator. Two lists are still used to differentiate between the traffic from the two trace files. The simulator compares well with the implementation of the non-preemptive simulator. The difference is that rather considering high priority packets that are waiting in the queue before considering low priority packets, all packets from both lists are considered equally. The simulator will choose the packet with the smallest absolute arrival time from the packets that are waiting in the list, rather than the high priority packet with the smallest absolute arrival time.

3.4 Queue Length

One performance characteristic that will be used to compare the simulators will be a graph that will show the queue length at different time intervals. The queue length can be calculated using the start and end times, absolute arrival times of the packets and a counter to represent the time instance where the queue length is evaluated. For each value of the counter, the list of absolute arrival times of the completed packets list is considered. If the current value of the counter is greater than the absolute arrival time, completion time of the packet is greater than the counter and the starting time of the packet is greater than the absolute arrival time of the packet, it means that the packet is waiting in the queue at the respective time the counter represents. This is done for all the packets at each time instance to determine how many packets are waiting in the queue at the time. The results are then plotted, with time on the x-axis and queue length on the y-axis.

3.5 Simulations

3.5.1 Given Trace Files

For the first test, each simulator was used to process two static trace files with fixed values. The trace files respectively include the low and high priority packets traffic. For the given trace files, each simulator will be used to calculate the average inter-arrival time, average transmission times, average response time, average queuing delay, average arrival rate and service rate. The queue length will also be plotted against time. This performance parameters will be calculated for the low and high priority packets separately, as well as the combined system performance parameters.

3.5.2 Averaged Results

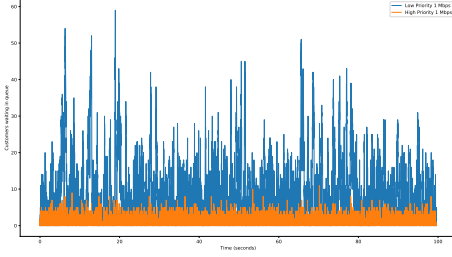
For this simulation, the same performance parameters will be calculated as in Section 3.5.1 and the queue length will also be plotted against time. The results will, however, be averaged out over 150 runs. The simulator will, therefore, generate the traces file 150 times, process the data and average the result. This will be done because arrival times and packet sizes are generated by using a random number generator and to get accurate behaviour of the system, an average of the results are required. The trace files will be generated with the same parameters that were calculated for the given trace files. We will see in section 4, that the given trace files yielded an average inter-arrival time of 2486 μs for the high priority packets and 1991 μs for the low priority packets. The number of high priority packets will be 40000 and 50000 for the low priority packets. The average packet size will be generated using a mean of 1000 bits per packet.

3.5.3 Varying Arrival Rates

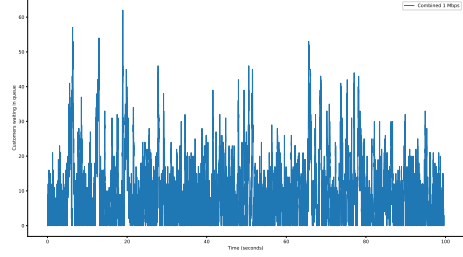
To compare what the effect of different arrival rates of the high priority packets are on the performance of the system, we will consider five different arrival rates for the high priority packets. The average will be taken of the results for each arrival rate as described in Subsection 3.5.2, but the results will be averaged over 100 runs. The arrival rates that will be considered for the high priority packets will be 2100, 2300, 2500, 2700 and 2900 microseconds.

4 Results

The first results that are provided will be for the given trace files for high and low priority packets. Table 2 shows the calculated performance parameters associated with each one of the simulators. The performance parameters were calculated for the high priority packets separately from the low priority packets to differentiate how the packets are serviced. The combined performance of high and low priority packets are also provided to indicate the total system performance. Average response time is calculated as the sum of the queuing time and the transmission time of a packet. The queuing delay is taken as the time that has elapsed from the point where the packet arrived, until the time the packet starts being serviced. As described in Section 3.4, the queue length will be calculated at different time instances for the trace files. This will be done for the non-preemptive, preemptive non-resume and the no priority simulator. For the provided trace files, the queue length was calculated for time intervals of ten milliseconds. These results are provided in Figure 2 to 4.

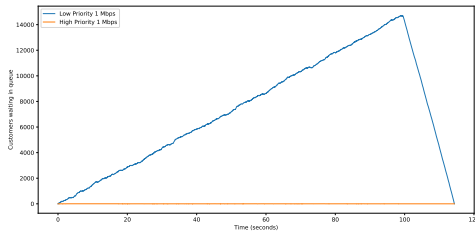


(a) High and low priority queues.

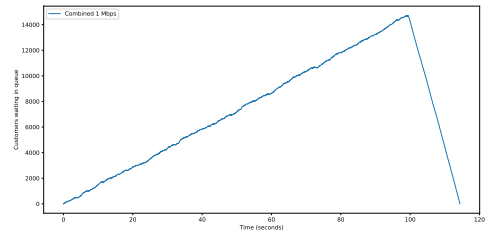


(b) Combined priorities.

Figure 2. High and low priority queue lengths plotted against time separately 2(a) and combined 2(b) for a 1 Mbps non-preemptive priority simulator.

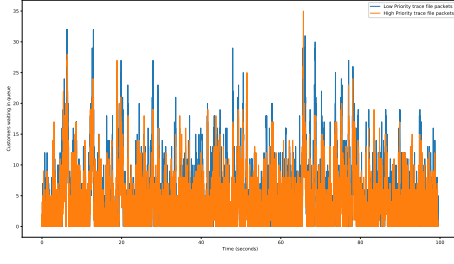


(a) High and low priority queues.

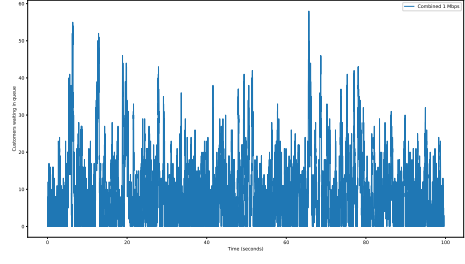


(b) Combined priorities.

Figure 3. High and low priority queue lengths plotted against time separately 3(a) and combined 3(b) for a 1 Mbps preemptive non-resume priority simulator.



(a) Separate file queue lengths.



(b) Combined file queue lengths.

Figure 4. Two files of packets generated with no priority. Queue lengths plotted against time separately 4(a) and combined 4(b) for a 1 Mbps no priority simulator.

The results provided in Table 2 and 3 are only for the trace files that were provided in this practical. Although the results of the provided trace files provide some insight into the performance of priority queuing, further results are needed to clarify confidence in the results obtained. Due to the randomly generated packet size and inter-arrival times, the results need to be averaged over various simulations to provide a better understanding of how the system performs under the defined input parameters. The results provided in Figures 5 to 7 are plotted in one-second intervals and averaged out over 150 simulations, as well as the results provided in Table 4.

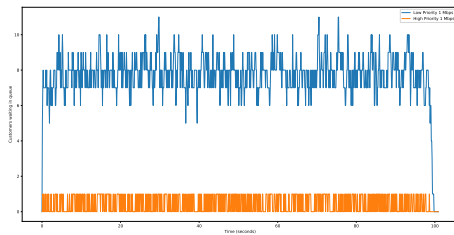
Simulator	Parameter	Calculated for packet priority			Unit
		Low	High	Combined	
Non-preemptive	Average response time	16340.423	2517.5917	10196.9424	μs
Preemptive		16727115.2507	1673.0544	9293585.3857	μs
No priority		10159.2036	10132.7128	10147.4299	μs
Non-preemptive	Average queueing delays	15338.0252	1514.4253	9194.203	μs
Preemptive		16726112.8529	669.888	9292582.6463	μs
No priority		9156.8058	9129.5464	9144.6905	μs
Non-preemptive	Queue length	8.1513	0.9781	9.1294	packets
Preemptive		6969.2104	0.2299	6969.4403	packets
No priority		5.0461	4.0394	9.0854	packets

Table 2. Provided trace file performance parameters calculated for a 1 Mbps link capacity with 50000 low priority packets and 40000 high priority packets.

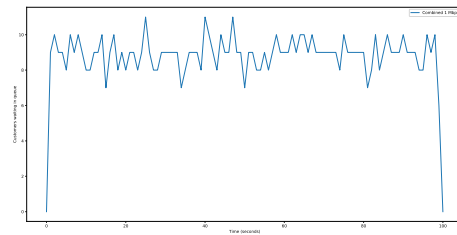
Table 3 shows the calculated average transmission time and service time. These parameters will stay consistent for the results that will be provided in the varying arrival rates, as well as the averaged results. This is due to packet size and link capacity that is fixed. The number of packets that will be generated will be determined based on the average inter-arrival times and the fact that approximately 100 seconds of traffic needs to be generated. The number of packets that will be generated with the various average inter-arrival times will be equal to 100 seconds divided by the average inter-arrival time.

Parameter	Low, high or combined priority	Value	Unit
Number of packets	Low	50000	packets
	High	40000	packets
	Combined	90000	packets
Average inter-arrival time	Low	1991.8206	μs
	High	2486.6197	μs
	Combined	2211.7313	μs
Average transmission time	Low	1002.3978	μs
	High	1003.1664	μs
	Combined	1002.7394	μs
Arrival rate	Low	502.0532	packets/s
	High	402.1524	packets/s
	Combined	904.2053	packets/s
Service rate	Low	997.6079	packets/s
	High	996.8436	packets/s
	Combined	997.2681	packets/s

Table 3. Performance parameters calculated for the given trace file.

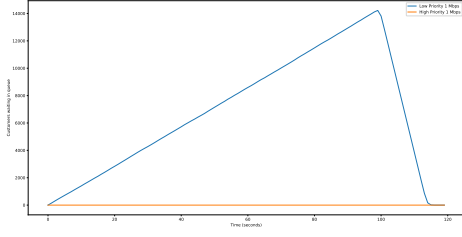


(a) High and low priority queues.

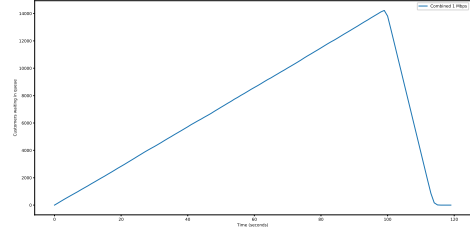


(b) Combined priorities.

Figure 5. High and low priority queue lengths plotted against time separately 5(a) and combined 5(b) for a 1 Mbps non-preemptive priority simulator, averaged over 150 simulations.

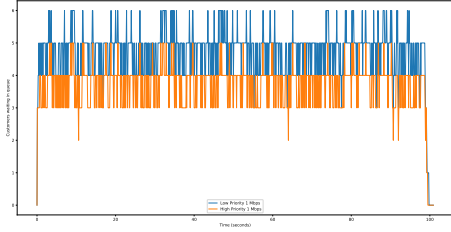


(a) High and low priority queues.

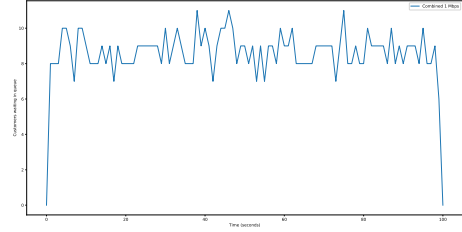


(b) Combined priorities.

Figure 6. High and low priority queue lengths plotted against time separately 6(a) and combined 6(b) for a 1 Mbps preemptive non-resume priority simulator, averaged over 150 simulations.



(a) Separate file queue lengths.



(b) Combined file queue lengths.

Figure 7. Two files of packets generated with no priority. Queue lengths plotted against time separately 7(a) and combined 7(b) for a 1 Mbps preemptive no priority simulator and averaged over 150 simulations.

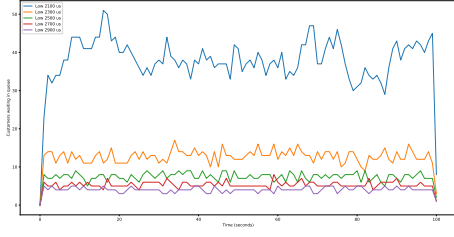
		Calculated for packet priority			Unit
Simulator	Parameter	Low	High	Combined	
Non-preemptive	Average response time	16807.1566	2511.3844	10453.48	μs
Preemptive		16520435.8291	1672.9732	9178763.4487	μs
No priority		10368.7603	10387.9942	10377.3087	μs
Non-preemptive	Average queueing delays	15807.0084	1511.5727	9453.4814	μs
Preemptive		16519435.2932	672.6596	9177763.0116	μs
No priority		9368.5959	9388.1642	9377.2929	μs
Non-preemptive	Average queue length	7.802	0.4554	8.7624	packets
Preemptive		6883.1167	0.0	6883.4	packets
No priority		4.5842	3.5149	8.5446	packets

Table 4. Averaged performance parameters over 150 simulations, calculated for a 1 Mbps link capacity with 50000 low priority packets and 40000 high priority packets.

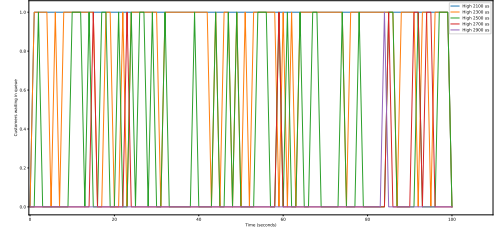
To verify what the effects of alteration of the arrival rate of the packets of different priorities are on the performance parameters, the average inter-arrival times of the high priority packets were varied from $2100\mu s$ to $2900\mu s$, in $200\mu s$ intervals. These results were also averaged out over 100 simulations. The results are provided in Figures 8 to 10 and Table 5.

Parameter	Simulator	IAT	Packet priority			Unit
			Low	High	Combined	
Average queuing delay	Non-preemptive	2100	81078	1864	42437	μs
		2300	25537	1654	14429	μs
		2500	14889	1496	8936.99	μs
		2700	10766	1384	6774	μs
		2900	8213	1286	5386	μs
	Preemptive	2100	28022574	908	14353475	μs
		2300	21222463	764	11351937	μs
		2500	16033262	666	8907664	μs
		2700	11617568	588	6674175	μs
		2900	7725751	526	4572639	μs
	No priority	2100	42152	42212	42181	μs
		2300	14155	14166	14160	μs
		2500	8889	8878	8884	μs
		2700	6694	6696	6695	μs
		2900	5366	5373	5369	μs
Average queue length	Non-preemptive	2100	37.76	0.9802	39.74	packets
		2300	12.60	0.7822	13.38	packets
		2500	7.37	0.3267	7.70	packets
		2700	5.28	0.0792	5.36	packets
		2900	4.0	0.0099	4.01	packets
	Preemptive	2100	11072	0.016	11072	packets
		2300	8488	0.00	8488	packets
		2500	6413	0.00	6413	packets
		2700	4646	0.00	4646	packets
		2900	3090	0.00	3090	packets
	No priority	2100	20.72	19.73	40.45	packets
		2300	7.0	5.98	12.98	packets
		2500	4.38	3.44	7.83	packets
		2700	3.23	2.26	5.50	packets
		2900	2.59	1.70	4.29	packets

Table 5. Averaged performance parameters over 100 simulations, calculated for a 1 Mbps link capacity with varying the average inter-arrival times (IAT) in microseconds.

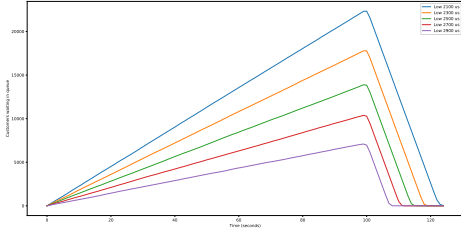


(a) Low priority queue length.

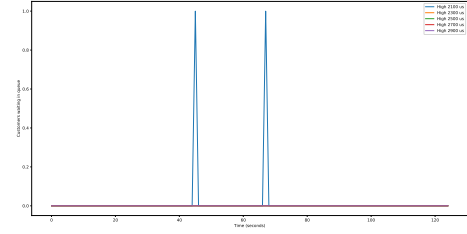


(b) High priority queue length.

Figure 8. Low 8(a) and high 8(b) priority queue lengths plotted against time for varying mean inter-arrival time of high priority packets for a 1 Mbps non-preemptive priority simulator, averaged over 100 simulations.

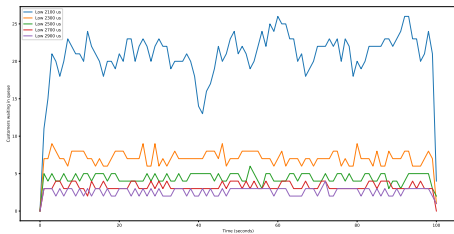


(a) Low priority queue length.

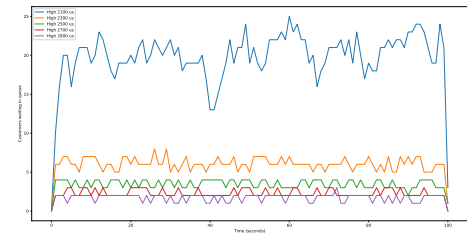


(b) High priority queue length.

Figure 9. Low 9(a) and high 9(b) priority queue lengths plotted against time for varying mean inter-arrival time of high priority packets for a 1 Mbps preemptive non-resume priority simulator, averaged over 100 simulations.



(a) First file queue length.



(b) Second queue length.

Figure 10. Two files of packets generated with no priority. Queue lengths plotted against time for first file 10(a) and second file 10(b) for a 1 Mbps no priority simulator and averaged over 100 simulations.

5 Discussion

After successful implementation of the preemptive non-resume, non-preemptive and no priority simulator as illustrated in Figure 1, the results that were obtained in Section 4 can be used to draw some conclusions of the performance of the simulators. For the provided trace files, we can get an overview of how each one of the simulators performs. For the non-preemptive priority simulator, we see from Figure 2(a) that the overall queue length for high priority packets is lower than low priority packets. It is seen that the low priority packet queue length does not increase as time progresses, but seems to behave similarly over time. The same is seen for the high priority packets, with the difference being that the average length of the queue seems to be shorter than for the case of low priority packets. We see that the total queue length of the system is shown in Figure 2(b), combining the low and high priority packet queue lengths. This simulation was done for an arrival rate of approximately 500 packets per second for the low priority packets and 400 packets per second for the high priority packets. Due to the arrival rate of low priority packets being higher than for the high priority packets, one would expect the queue length for low priority packets to be longer than the high priority packets. That said, the advantage of high priority packets being serviced before low priority packets would further clarify why the queue length for high priority packets is lower than for low priority packets. We see in Table 3 that the transmission times and the service rates of low and high priority packets are approximately the same because transmission time is dependent on the link capacity and the average packet size, which are the same for both high and low priority packets. Figure 3 shows the queue length of high and low priority packets for the preemptive non-resume priority simulator. We see from Figure 3(a) that the queue length of low priority packets keeps increasing up to the 100-second point where no traffic is received. This is a concern to the system, as the queue length of the system will keep increasing as traffic is received. This may cause that low priority packets experience very long queuing delays, as time progresses and traffic is still arriving at the specified rates. This is, however, not a problem for high priority packets, as we see that the queue length of high priority packets is approximately zero at all times and high priority packets almost immediately receive service. From Figure 4, the results are shown of the no priority simulator. We see that the queue lengths of the file that generates packets at a rate of 400 packets per second are shorter than the queue lengths of file generating packets at a rate of 500 packets per second. Comparing these results to Figure 2, we see that for non-preemptive priority, the high priority packets receive service faster than low priority packets. The total combined queue lengths are, however, similar for the no priority and non-preemptive simulators as seen in Figure 2(b) and 4(b). This is seen in Table 2, where the combined priority queuing delays, response time and queue lengths are approximately the same. Compared to the preemptive non-resume priority simulator results in Table 2 and Figure 3(b), we see that the average queue length, queuing delays and response time for the total system packets performs much worse than for the non-preemptive and no priority simulators. The advantage of the preemptive non-resume priority is that high priority packets are serviced much faster and experience much shorter queues.

From Figure 5 to 7, the average results were obtained for each of the simulators over

150 runs. The average inter-arrival times and packet sizes were generated using an exponential distribution to simulate the Poisson arrival and service process. An arrival rate of 400 packets per second was used for high priority packets and 500 packets per second were used for low priority packets. An average packet size of 1000 bits was used for both high and low priority packets with a link capacity of 1 Mbps. From the averaged results in Figures 5 and 7, we can get a better graphical representation of the average queue length of high and low priority packets. These results are very similar compared to the results obtained in the trace files, comparing Table 2 and 4 and we can, therefore, confirm the observations we made for the trace file results.

To see what the effect of different arrival rates of high priority packets have on the performance of the system, the average inter-arrival times that were generated in the trace files for the high priority packets was changed from $2100\mu s$ to $2900\mu s$, in $200\mu s$ intervals. The results are shown in Figures 8 to 10 and in Table 5. From Figure 8(a) we see that the average queue length of the low priority packets decreases as the arrival rate of high priority packets decreases. We can also see that the average queue length of the low priority packets seems to stay the same for all times, as the arrival rates are fixed. Figure 8(b) shows that for the chosen arrival rates of the high priority packets, the maximum queue length will only be one, showing that high priority packets are quickly serviced. Figure 9(a) shows that the low priority queue lengths keep growing as time progresses for all the chosen arrival rates of high priority packets. It would be advised that this priority scheme should only be used with very low arrival rates for the high priority packets to prevent the queue length of the system growing infinite over time. The arrival rate of the high priority packets will need to be low enough to ensure that the point is reached where sufficient low priority packets receive service for long enough to complete and leave the system. This will ensure that the number of completions is greater or approximately the same as the number of arrivals, causing queue lengths and queuing delays to be limited to reasonable lengths and times. Figure 9(b) illustrates that the queue length of high priority packets are mostly zero for all the arrival rates chosen for the high priority packets and we can further lower the arrival rate for high priority packets for reasonable queue lengths for the high priority packets, allowing low priority packets to receive better service. As one expects for the case where no priority is assigned to packets, Figures 10(a) and 10(b) illustrates that the queue lengths are approximately the same for both trace files generating the traffic.

From the results and observations made, we can say that for the non-preemptive simulator, queue lengths in the system are consistent throughout time and are reasonable. The overall system performance is similar to the case where no priorities are assigned, with high priority packets receiving better service. For the preemptive non-resume simulations, we saw that the system may become unstable, with very high waiting times, queue lengths and response times as time progresses and traffic is received continuously. Care has to be taken in this case to limit the arrival rate of high priority packets to the point where enough low priority packets receive uninterrupted service to be completed. This will ensure the system to remain stable and queue lengths and waiting times to not tend toward infinite.

6 Conclusion

In light of the content of this report, it was seen that the preemptive non-resume, non-preemptive and no priority simulators were implemented successfully. The simulators handled two priorities of packets. It was seen that with the preemptive non-resume priority simulator, the high priority packets got serviced at a very high rate, with queue lengths averaging approximately zero for the high priority packets at the considered arrival rates of the high and low priority packets. The problem with preemptive non-resume priority scheduling is that the queuing delays and queue lengths for the low priority packets and the system as a whole are very high if time increases. Care should be taken to limit the arrival rate of high priority packets to the point where low priority packets are allowed to be serviced to completion for system stability. If this is not done, we saw that the queuing lengths and queuing delays will keep growing as time progresses if the system keeps receiving packets at the specified arriving rates. For non-preemptive and no priority scheduling, we saw that the total system performance were approximately the same, while the high priority packets received faster service than low priority packets when the non-preemptive priority simulator was used. Queue length and queue delay reaches a constant averaged value for non-preemptive priority scheduling and can be controlled by limiting the arrival rate of high priority packets. It will be advised that non-preemptive scheduling should be used in cases with a high arrival rate of high priority packets, and preemptive non-resume priority scheduling should be used in cases where the arrival rate of high priority packets is low enough for system stability. No priority scheduling showed equivalent performances for all packets and can be used in cases where all customers should be treated equally.

7 References

- [1] L. Strydom, *ERP 420 – Practical 2 Priority Queuing and Queuing Management Systems v1.1*, University of Pretoria, 8 Sept. 2019.
- [2] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains*, 2nd ed. Wiley, 2006.

A Simulators Python Souce Code

```
1  # -*- coding: utf-8 -*-
   """
3  Created on Mon Aug  5 15:15:31 2019

5  @author: project
   """

7
9  import csv
10 import time

11
12 class Packet:
13     def __init__(self, size, interArrivalTime, priority):
14         self.size = size
15         self.interArrivalTime = interArrivalTime
16         self.priority = priority
17         self.arrivalTime = 0
18         self.transTime = 0
19         self.queueDelay = 0
20         self.startTime = 0
21         self.endTime = 0
22         self.responseTime = 0
23
24 # Read the text file and the inter-arrival time and the size
25 def readCSV(filePath):
26     arrival = []
27     size = []
28
29     with open(filePath, 'r') as f:
30         data = csv.reader(f)
31         for row in data:
32             arrival.append(row[0])
33             size.append(row[1])
34
35 # Convert the strings lists to integer lists
36 arrival = list(map(int, arrival))
37 size = list(map(int, size))
38
39 return arrival, size
40
41
42 def getTraceInfo():
43     # Read the csv file data into respective lists to store the data
44     interArrivalHigh, packetSizeHigh = readCSV('HighPriority.txt')
45     interArrivalLow, packetSizeLow = readCSV('LowPriority.txt')
46     print("Low Priority Packets :", len(interArrivalLow))
47     print("High Priority Packets :", len(interArrivalHigh))
48     print("Avg interarrival Low :", sum(interArrivalLow)/len(
49 interArrivalLow))
50     print("Avg interarrival High :", sum(interArrivalHigh)/len(
51 interArrivalHigh))
52     print("Avg size Low           :", sum(packetSizeLow)/len(packetSizeLow
53 ))
```

```

51     print("Avg size High      :", sum(packetSizeHigh)/len(
        packetSizeHigh))
        print("Total interarrival      :", (sum(interArrivalHigh)+sum(
            interArrivalLow))/(len(interArrivalHigh)+len(interArrivalLow)))
53
55 getTraceInfo()

57 # Takes the packets as a list of two lists corresponding to
    # [[class:packetLowPriority], class[packetHighPriority]] and the
        transmission
59 # capacity as an integer and returns arrivalTimes, transTimes,
        queuingDelays,
    # startEnd, idleTimes, respTimes.
61 def calcNonPreemptive(transCap, packets):
    currTime = 0
63     completedPackets = []

65     # Calculate when the packets were received for low priority
    for i in range(0, len(packets[0])):
67         if (i == 0):
            packets[0][i].arrivalTime = packets[0][i].interArrivalTime
69
            else:
71                 packets[0][i].arrivalTime = packets[0][i].interArrivalTime +
                    packets[0][i - 1].arrivalTime

73     # Calculate when the packets were received for high priority
    for i in range(0, len(packets[1])):
75         if (i == 0):
            packets[1][i].arrivalTime = packets[1][i].interArrivalTime
77
            else:
79                 packets[1][i].arrivalTime = packets[1][i].interArrivalTime +
                    packets[1][i - 1].arrivalTime

81     # Calculate how long each packet takes to transmit for low
        priority
    for i in range(0, len(packets[0])):
83         packets[0][i].transTime = (packets[0][i].size/transCap)*10**6

85     # Calculate how long each packet takes to transmit for high priority
    for i in range(0, len(packets[1])):
87         packets[1][i].transTime = (packets[1][i].size/transCap)*10**6

89
91     while (len(packets[0]) > 0 or len(packets[1]) > 0):
        # For the first packet arriving

93         toProcess = None

95         # If a high priority packet was received
        if (len(packets[1]) > 0 and packets[1][0].arrivalTime <= currTime
97     ):
        toProcess = packets[1][0]
        packets[1] = packets[1][1:]

```

```

99
101     # If a low priority packet was received and no high priority
    elif (len(packets[0]) > 0 and packets[0][0].arrivalTime <=
currTime):
103         toProcess = packets[0][0]
        packets[0] = packets[0][1:]

105     # If server is idle , get the next packet and update the packet to
    # the corresponding arrival time
107     else:
        # If there are still packets to be processed of high and low
priority
109         if (len(packets[1]) > 0 and len(packets[0]) > 0):
            if (packets[1][0].arrivalTime <= packets[0][0].
arrivalTime):
111                 toProcess = packets[1][0]
                    currTime = packets[1][0].arrivalTime
113                 packets[1] = packets[1][1:]

115             else:
                toProcess = packets[0][0]
                currTime = packets[0][0].arrivalTime
                packets[0] = packets[0][1:]

119             elif (len(packets[1]) > 0):
                toProcess = packets[1][0]
                currTime = packets[1][0].arrivalTime
                packets[1] = packets[1][1:]

123             elif (len(packets[0]) > 0):
                toProcess = packets[0][0]
                currTime = packets[0][0].arrivalTime
                packets[0] = packets[0][1:]

129         if (toProcess != None):
131             # Process all the packets that has arrived
            toProcess.startTime = currTime
            toProcess.endTime = currTime + toProcess.transTime
            toProcess.queueDelay = currTime - toProcess.arrivalTime
            completedPackets.append(toProcess)
            currTime = toProcess.endTime
            toProcess = None

139     # Calculate the reponse times for each of the packets
    for i in range(len(completedPackets)):
141         completedPackets[i].responseTime = completedPackets[i].endTime -
completedPackets[i].startTime + completedPackets[i].queueDelay

143     arrivalTimes = [[], []]
    transTimes = [[], []]
145     queuingDelays = [[], []]
    startEnd = [[], []]
147     respTimes = [[], []]

149     for i in range(0, len(completedPackets)):

```

```

151         if (completedPackets[i].priority == "HIGH"):
152             arrivalTimes[1].append(completedPackets[i].arrivalTime)
153             transTimes[1].append(completedPackets[i].transTime)
154             queuingDelays[1].append(completedPackets[i].queueDelay)
155             startEnd[1].append([completedPackets[i].startTime,
completedPackets[i].endTime])
156             respTimes[1].append(completedPackets[i].responseTime)
157
158         elif (completedPackets[i].priority == "LOW"):
159             arrivalTimes[0].append(completedPackets[i].arrivalTime)
160             transTimes[0].append(completedPackets[i].transTime)
161             queuingDelays[0].append(completedPackets[i].queueDelay)
162             startEnd[0].append([completedPackets[i].startTime,
completedPackets[i].endTime])
163             respTimes[0].append(completedPackets[i].responseTime)
164
165     return arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
completedPackets
166
167 # Takes the packets as a list of two lists corresponding to
168 # [[class:packetLowPriority], class[packetHighPriority]] and the
169 # transmission
170 # capacity as an integer and returns arrivalTimes, transTimes,
171 # queuingDelays,
172 # startEnd, idleTimes, respTimes.
173 def calcPreemptive(transCap, packets):
174     currTime = 0
175     completedPackets = []
176
177     # Calculate when the packets were received for low priority
178     for i in range(0, len(packets[0])):
179         if (i == 0):
180             packets[0][i].arrivalTime = packets[0][i].interArrivalTime
181
182         else:
183             packets[0][i].arrivalTime = packets[0][i].interArrivalTime +
184             packets[0][i - 1].arrivalTime
185
186     # Calculate when the packets were received for high priority
187     for i in range(0, len(packets[1])):
188         if (i == 0):
189             packets[1][i].arrivalTime = packets[1][i].interArrivalTime
190
191         else:
192             packets[1][i].arrivalTime = packets[1][i].interArrivalTime +
193             packets[1][i - 1].arrivalTime
194
195     # Calculate how long each packet takes to transmit for low priority
196     for i in range(0, len(packets[0])):
197         packets[0][i].transTime = (packets[0][i].size/transCap)*10**6
198
199     # Calculate how long each packet takes to transmit for high priority
200     for i in range(0, len(packets[1])):
201         packets[1][i].transTime = (packets[1][i].size/transCap)*10**6
202
203     while (len(packets[0]) > 0 or len(packets[1]) > 0):

```

```

201     # For the first packet arriving
    toProcess = None

203     # If a high priority packet was received
    if (len(packets[1]) > 0 and packets[1][0].arrivalTime <= currTime
):
205         toProcess = packets[1][0]
        packets[1] = packets[1][1:]

207     # If a low priority packet was received and no high priority
    elif (len(packets[0]) > 0 and packets[0][0].arrivalTime <=
209 currTime):
        toProcess = packets[0][0]

211     # If server is idle, get the next packet and update the packet to
213     # the corresponding arrival time
    else:
215         # If there are still packets to be processed of high and low
        priority
        if (len(packets[1]) > 0 and len(packets[0]) > 0):
217             if (packets[1][0].arrivalTime <= packets[0][0].
arrivalTime):
                toProcess = packets[1][0]
                currTime = packets[1][0].arrivalTime
                packets[1] = packets[1][1:]

221             else:
                toProcess = packets[0][0]
                currTime = packets[0][0].arrivalTime

223         elif (len(packets[1]) > 0):
            toProcess = packets[1][0]
            currTime = packets[1][0].arrivalTime
            packets[1] = packets[1][1:]

229         elif (len(packets[0]) > 0):
            toProcess = packets[0][0]
            currTime = packets[0][0].arrivalTime

235     if (toProcess != None):
        toProcess.startTime = currTime
        toProcess.endTime = currTime + toProcess.transTime
        toProcess.queueDelay = currTime - toProcess.arrivalTime
        currTime = toProcess.endTime

241     # Process all the packets that has arrived
    if (toProcess.priority == "LOW"):
243         if (len(packets[1]) > 0 and currTime > packets[1][0].
arrivalTime):
            currTime = packets[1][0].arrivalTime
            toProcess = None
            continue

247         else:
            packets[0] = packets[0][1:]

```

```

251         completedPackets.append(toProcess)
253
254     # Calculate the reponse times for each of the packets
255     for i in range(len(completedPackets)):
256         completedPackets[i].responseTime = completedPackets[i].endTime -
257         completedPackets[i].startTime + completedPackets[i].queueDelay
258
259     arrivalTimes = [[], []]
260     transTimes = [[], []]
261     queuingDelays = [[], []]
262     startEnd = [[], []]
263     respTimes = [[], []]
264
265     for i in range(0, len(completedPackets)):
266         if (completedPackets[i].priority == "HIGH"):
267             arrivalTimes[1].append(completedPackets[i].arrivalTime)
268             transTimes[1].append(completedPackets[i].transTime)
269             queuingDelays[1].append(completedPackets[i].queueDelay)
270             startEnd[1].append([completedPackets[i].startTime,
271             completedPackets[i].endTime])
272             respTimes[1].append(completedPackets[i].responseTime)
273
274             elif (completedPackets[i].priority == "LOW"):
275                 arrivalTimes[0].append(completedPackets[i].arrivalTime)
276                 transTimes[0].append(completedPackets[i].transTime)
277                 queuingDelays[0].append(completedPackets[i].queueDelay)
278                 startEnd[0].append([completedPackets[i].startTime,
279                 completedPackets[i].endTime])
280                 respTimes[0].append(completedPackets[i].responseTime)
281
282     return arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
283     completedPackets
284
285 # Takes the packets as a list of two lists corrensponding to
286 # [[class:packetLowPriority], class[packetHighPriority]] and the
287 # transmission
288 # capacity as an integer and returns arrivalTimes, transTimes,
289 # queuingDelays,
290 # startEnd, idleTimes, respTimes.
291 def calcNoPriority(transCap, packets):
292
293     currTime = 0
294     completedPackets = []
295
296     # Calculate when the packets were received for low priority
297     for i in range(0, len(packets[0])):
298         if (i == 0):
299             packets[0][i].arrivalTime = packets[0][i].interArrivalTime
300
301         else:
302             packets[0][i].arrivalTime = packets[0][i].interArrivalTime +
303             packets[0][i - 1].arrivalTime
304
305     # Calculate when the packets were received for high priority

```

```

301     for i in range(0, len(packets[1])):
302         if (i == 0):
303             packets[1][i].arrivalTime = packets[1][i].interArrivalTime
304
305         else:
306             packets[1][i].arrivalTime = packets[1][i].interArrivalTime +
307             packets[1][i - 1].arrivalTime
308
309     # Calculate how long each packet takes to transmit for low priority
310     for i in range(0, len(packets[0])):
311         packets[0][i].transTime = (packets[0][i].size/transCap)*10**6
312
313     # Calculate how long each packet takes to transmit for high priority
314     for i in range(0, len(packets[1])):
315         packets[1][i].transTime = (packets[1][i].size/transCap)*10**6
316
317     while (len(packets[0]) > 0 or len(packets[1]) > 0):
318         # For the first packet arriving
319         toProcess = None
320
321         if (len(packets[1]) > 0 and len(packets[0]) > 0 and packets
322             [1][0].arrivalTime <= currTime and packets[0][0].arrivalTime <=
323             currTime):
324             if (packets[1][0].arrivalTime <= packets[0][0].arrivalTime):
325                 toProcess = packets[1][0]
326                 packets[1] = packets[1][1:]
327
328             else:
329                 toProcess = packets[0][0]
330                 packets[0] = packets[0][1:]
331
332             # If a high priority packet was received
333             elif (len(packets[1]) > 0 and packets[1][0].arrivalTime <=
334                 currTime):
335                 toProcess = packets[1][0]
336                 packets[1] = packets[1][1:]
337
338             # If a low priority packet was received and no high priority
339             elif (len(packets[0]) > 0 and packets[0][0].arrivalTime <=
340                 currTime):
341                 toProcess = packets[0][0]
342                 packets[0] = packets[0][1:]
343
344             # If server is idle, get the next packet and update the packet to
345             # the corresponding arrival time
346             else:
347                 # If there are still packets to be processed of high and low
348                 priority
349                 if (len(packets[1]) > 0 and len(packets[0]) > 0):
350                     if (packets[1][0].arrivalTime < packets[0][0].arrivalTime
351                         ):
352                         toProcess = packets[1][0]
353                         currTime = packets[1][0].arrivalTime
354                         packets[1] = packets[1][1:]

```

```

349         else:
350             toProcess = packets[0][0]
351             currTime = packets[0][0].arrivalTime
352             packets[0] = packets[0][1:]
353
354         elif (len(packets[1]) > 0):
355             toProcess = packets[1][0]
356             currTime = packets[1][0].arrivalTime
357             packets[1] = packets[1][1:]
358
359         elif (len(packets[0]) > 0):
360             toProcess = packets[0][0]
361             currTime = packets[0][0].arrivalTime
362             packets[0] = packets[0][1:]
363
364     if (toProcess != None):
365         # Process all the packets that has arrived
366         toProcess.startTime = currTime
367         toProcess.endTime = currTime + toProcess.transTime
368         toProcess.queueDelay = currTime - toProcess.arrivalTime
369         completedPackets.append(toProcess)
370         currTime = toProcess.endTime
371         toProcess = None
372
373     # Calculate the reponse times for each of the packets
374     for i in range(len(completedPackets)):
375         completedPackets[i].responseTime = completedPackets[i].endTime -
completedPackets[i].startTime + completedPackets[i].queueDelay
376
377     arrivalTimes = [[], []]
378     transTimes = [[], []]
379     queuingDelays = [[], []]
380     startEnd = [[], []]
381     respTimes = [[], []]
382
383
384     for i in range(0, len(completedPackets)):
385         if (completedPackets[i].priority == "HIGH"):
386             arrivalTimes[1].append(completedPackets[i].arrivalTime)
387             transTimes[1].append(completedPackets[i].transTime)
388             queuingDelays[1].append(completedPackets[i].queueDelay)
389             startEnd[1].append([completedPackets[i].startTime,
completedPackets[i].endTime])
390             respTimes[1].append(completedPackets[i].responseTime)
391
392         elif (completedPackets[i].priority == "LOW"):
393             arrivalTimes[0].append(completedPackets[i].arrivalTime)
394             transTimes[0].append(completedPackets[i].transTime)
395             queuingDelays[0].append(completedPackets[i].queueDelay)
396             startEnd[0].append([completedPackets[i].startTime,
completedPackets[i].endTime])
397             respTimes[0].append(completedPackets[i].responseTime)
398
399     return arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
completedPackets

```


B Trace File Driver

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Sep 16 09:40:23 2019
4
5 @author: project
6 """
7
8 import matplotlib.pyplot as plt
9 from PriorityQueuing import readCSV, Packet, calcNoPriority,
10     calcNonPreemptive, calcPreemptive
11 import numpy as np
12 import time as t
13 from operator import add
14
15 doNonPreemptiveTrace = True
16 doPreemptiveTrace = False
17 doNoPriority = False
18
19 linkCapacity = 1*10**6
20
21 if (doNoPriority == True):
22     print("
23     #####
24     ##### Processing no priority queuing
25     #####")
26     print("
27     #####")
28
29     packets = [[], []]
30     time = []
31
32     transCap = linkCapacity
33
34     # Read the csv file data into respective lists to store the data
35     interArrivalHigh, packetSizeHigh = readCSV('HighPriority.txt')
36     interArrivalLow, packetSizeLow = readCSV('LowPriority.txt')
37
38     # Create the list of low priority packets
39     for i in range(0, len(interArrivalLow)):
40         lowPacket = Packet(packetSizeLow[i], interArrivalLow[i], 'LOW')
41         packets[0].append(lowPacket)
42
43     # Create the list of high priority packets
44     for i in range(0, len(interArrivalHigh)):
```

```

43         highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH
    ')
44         packets[1].append(highPacket)
45
46     # Get the parameters
47     arrivalTimes, transTimes, queuingDelays, startEnd, respTimes, allPackets =
        calcNoPriority(transCap, packets)
48     sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))), float(
        sum(packetSizeHigh))/len(packetSizeHigh)]
49
50     print("Link capacity -----", transCap/10**6, "
    Mbps")
51     print("Number of low priority packets -----", len(packetSizeLow),
        "packets")
52     print("Number of high priority packets -----", len(packetSizeHigh)
        , "packets")
53     print("Total number of packets -----", len(packetSizeLow)
        + len(packetSizeHigh), "packets")
54     print("Average inter-arrival time -- (LOW) ----", round((sum(
        interArrivalLow))/len(interArrivalLow), 4), "us")
55     print("Average inter-arrival time -- (HIGH) ----", round((sum(
        interArrivalHigh))/len(interArrivalHigh), 4), "us")
56     print("Average inter-arrival time -- (TOTAL) --", round((sum(
        interArrivalHigh) + sum(interArrivalLow))/len(interArrivalHigh) + len
        (interArrivalLow)), 4), "us")
57     print("Average transmission time --- (LOW) ----", round(sum(
        transTimes[0])/len(transTimes[0]), 4), "us")
58     print("Average transmission time --- (HIGH) ----", round(sum(
        transTimes[1])/len(transTimes[1]), 4), "us")
59     print("Average transmission time --- (TOTAL) --", round((sum(
        transTimes[1]) + sum(transTimes[0]))/(len(transTimes[1])+len(
        transTimes[0])), 4), "us")
60     print("Average response time ----- (LOW) ----", round(sum(respTimes
        [0])/len(respTimes[0]), 4), "us")
61     print("Average response time ----- (HIGH) ----", round(sum(respTimes
        [1])/len(respTimes[1]), 4), "us")
62     print("Average response time ----- (TOTAL) --", round((sum(
        respTimes[0]) + sum(respTimes[1]))/(len(respTimes[0]) + len(respTimes
        [1])), 4), "us")
63     print("The average packet size ----- (LOW) ----", round(sizeAvg[0],
        4), "bits")
64     print("The average packet size ----- (HIGH) ----", round(sizeAvg[1],
        4), "bits")
65     print("The average packet size ----- (TOTAL) --", round((sum(
        packetSizeLow) + sum(packetSizeHigh)) / (float(len(packetSizeLow)) +
        float(len(packetSizeHigh))), 4), "bits")
66     print("Average delays ----- (LOW) ----", round(sum(
        queuingDelays[0])/len(queuingDelays[0]), 4), "us")
67     print("Average delays ----- (HIGH) ----", round(sum(
        queuingDelays[1])/len(queuingDelays[1]), 4), "us")
68     print("Average delays ----- (TOTAL) --", round((sum(
        queuingDelays[0]) + sum(queuingDelays[1]))/(len(queuingDelays[0]) +
        len(queuingDelays[1])), 4), "us")
69     print("Average arrival rate (lambda) (LOW) ----", round((len(
        packetSizeLow))/((sum(interArrivalLow))*10**6, 4), "packets/second")

```

```

    print("Average arrival rate (lambda) (HIGH) ---", round((len(
packetSizeHigh))/((sum(interArrivalHigh))*10**6, 4), "packets/second"
)
71     print("Average arrival rate (lambda) (TOTAL) --", round((len(
packetSizeLow) + len(packetSizeHigh))/(((sum(interArrivalHigh)*40000 +
sum(interArrivalLow)*50000)/90000))*10**6, 4), "packets/second")
    print("Average service rate (mu) --- (LOW) ----", round(transCap/
sizeAvg[0], 4), "packets/second")
73     print("Average service rate (mu) --- (HIGH) ----", round(transCap/
sizeAvg[1], 4), "packets/second")
    print("Average service rate (mu) --- (TOTAL) --", round(transCap/(((
sizeAvg[0]*50000 + sizeAvg[1]*40000)/90000)), 4), "packets/second")
75
    startEnd = [[], []]
    arrivalTimes = []
77     startEndLow = [[], []]
    arrivalTimesLow = []
79     startEndHigh = [[], []]
    arrivalTimesHigh = []
81
83     for i in allPackets:
        startEnd[0].append(i.startTime)
85         startEnd[1].append(i.endTime)
        arrivalTimes.append(i.arrivalTime)
87
        if (i.priority == 'LOW'):
89             startEndLow[0].append(i.startTime)
            startEndLow[1].append(i.endTime)
91             arrivalTimesLow.append(i.arrivalTime)
93
            if (i.priority == 'HIGH'):
                startEndHigh[0].append(i.startTime)
                startEndHigh[1].append(i.endTime)
                arrivalTimesHigh.append(i.arrivalTime)
97
99     time = np.arange(0, startEnd[1][-1], 10000)
    combined = []
    lowPriority = []
101    highPriority = []
103
    for i in range(0, len(time)):
        length = 0
105        for j in range(0, len(arrivalTimesLow)):
            if (time[i] >= arrivalTimesLow[j] and startEndLow[1][j] >
time[i]):
107                if (startEndLow[0][j] > arrivalTimesLow[j]):
                    length += 1
109                lowPriority.append(length)
111
    for i in range(0, len(time)):
        length = 0
113        for j in range(0, len(arrivalTimesHigh)):
            if (time[i] >= arrivalTimesHigh[j] and startEndHigh[1][j] >
time[i]):
115                if (startEndHigh[0][j] > arrivalTimesHigh[j]):
                    length += 1

```

```

117         highPriority.append(length)

119     combined = list( map(add, highPriority, lowPriority) )

121     print("Average queue length (LOW) -----", round(sum(
lowPriority)/len(lowPriority), 4), "packets/second")
    print("Average queue length (HIGH) -----", round(sum(
highPriority)/len(highPriority), 4), "packets/second")
123     print("Average queue length (TOTAL) -----", round(sum(combined
)/len(combined), 4), "packets/second")

125

127     time = time/10**6

129     plt.figure()
    plt.plot(time, combined, label=str('Combined 1 Mbps'))
    plt.legend(loc='best')
131     plt.xlabel('Time (seconds)')
    plt.ylabel('Customers waiting in queue')
133     plt.title('No priority combined Trace')
    plt.show()

135

137     plt.figure()
    plt.plot(time, lowPriority, label=str('Low Priority trace file
packets'))
    plt.plot(time, highPriority, label=str('High Priority trace file
packets'))
139     plt.legend(loc='best')
    plt.xlabel('Time (seconds)')
141     plt.ylabel('Customers waiting in queue')
    plt.title('No priority low and high Trace')
143     plt.show()

145

147 if (doNonPreemptiveTrace == True):
    print("
#####")
    print("##### Processing non-preemptive queuing
#####")
149     print("
#####")
    timeNow = t.time()

151

153     packets = [[],[]]
    time = []

155

157     transCap = linkCapacity

159     # Read the csv file data into respective lists to store the data
    interArrivalHigh, packetSizeHigh = readCSV('HighPriority.txt')
    interArrivalLow, packetSizeLow = readCSV('LowPriority.txt')

161     # Create the list of low priority packets
    for i in range(0, len(interArrivalLow)):
163         lowPacket = Packet(packetSizeLow[i], interArrivalLow[i], 'LOW')
        packets[0].append(lowPacket)

```

```

165 # Create the list of high priority packets
167 for i in range(0, len(interArrivalHigh)):
    highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH
    ')
169     packets[1].append(highPacket)

171 # Get the parameters
    arrivalTimes, transTimes, queuingDelays, startEnd, respTimes, allPackets =
        calcNonPreemptive(transCap, packets)
173     sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))), float(
        sum(packetSizeHigh))/len(packetSizeHigh)]

175     print("Link capacity -----", transCap/10**6, "
    Mbps")
    print("Number of low priority packets -----", len(packetSizeLow),
        "packets")
177     print("Number of high priority packets -----", len(packetSizeHigh)
        , "packets")
    print("Total number of packets -----", len(packetSizeLow)
    + len(packetSizeHigh), "packets" )
179     print("Average inter-arrival time -- (LOW) ----", round((sum(
        interArrivalLow))/(len(interArrivalLow)), 4), "us" )
    print("Average inter-arrival time -- (HIGH) ----", round((sum(
        interArrivalHigh))/(len(interArrivalHigh)), 4), "us" )
181     print("Average inter-arrival time -- (TOTAL) --", round((sum(
        interArrivalHigh) + sum(interArrivalLow))/(len(interArrivalHigh) + len
        (interArrivalLow)), 4), "us" )
    print("Average transmission time --- (LOW) ----", round(sum(
        transTimes[0])/len(transTimes[0]), 4), "us" )
183     print("Average transmission time --- (HIGH) ----", round(sum(
        transTimes[1])/len(transTimes[1]), 4), "us" )
    print("Average transmission time --- (TOTAL) --", round((sum(
        transTimes[1]) + sum(transTimes[0]))/(len(transTimes[1])+len(
        transTimes[0])), 4), "us" )
185     print("Average response time ----- (LOW) ----", round(sum(respTimes
        [0])/len(respTimes[0]), 4), "us" )
    print("Average response time ----- (HIGH) ----", round(sum(respTimes
        [1])/len(respTimes[1]), 4), "us" )
187     print("Average response time ----- (TOTAL) --", round((sum(
        respTimes[0]) + sum(respTimes[1]))/(len(respTimes[0]) + len(respTimes
        [1])), 4), "us" )
    print("The average packet size ----- (LOW) ----", round(sizeAvg[0],
        4), "bits" )
189     print("The average packet size ----- (HIGH) ----", round(sizeAvg[1],
        4), "bits" )
    print("The average packet size ----- (TOTAL) --", round((sum(
        packetSizeLow) + sum(packetSizeHigh)) / (float(len(packetSizeLow)) +
        float(len(packetSizeHigh))), 4), "bits" )
191     print("Average delays ----- (LOW) ----", round(sum(
        queuingDelays[0])/len(queuingDelays[0]), 4), "us" )
    print("Average delays ----- (HIGH) ----", round(sum(
        queuingDelays[1])/len(queuingDelays[1]), 4), "us" )
193     print("Average delays ----- (TOTAL) --", round((sum(
        queuingDelays[0]) + sum(queuingDelays[1]))/(len(queuingDelays[0]) +
        len(queuingDelays[1])), 4), "us" )

```

```

195     print("Average arrival rate (lambda) (LOW) ----", round((len(
        packetSizeLow))/((sum(interArrivalLow))*10**6, 4), "packets/second")
        print("Average arrival rate (lambda) (HIGH) ---", round((len(
        packetSizeHigh))/((sum(interArrivalHigh))*10**6, 4), "packets/second"
        )
        print("Average arrival rate (lambda) (TOTAL) --", round((len(
        packetSizeLow) + len(packetSizeHigh))/(((sum(interArrivalHigh)*40000 +
        sum(interArrivalLow)*50000)/90000))*10**6, 4), "packets/second")
197     print("Average service rate (mu) --- (LOW) ----", round(transCap/
        sizeAvg[0], 4), "packets/second")
        print("Average service rate (mu) --- (HIGH) ---", round(transCap/
        sizeAvg[1], 4), "packets/second")
199     print("Average service rate (mu) --- (TOTAL) --", round(transCap/(((
        sizeAvg[0]*50000 + sizeAvg[1]*40000)/90000)), 4), "packets/second")

201
202     startEnd = [[],[]]
203     arrivalTimes = []
        startEndLow = [[],[]]
205     arrivalTimesLow = []
        startEndHigh = [[],[]]
207     arrivalTimesHigh = []

209     for i in allPackets:
        startEnd[0].append(i.startTime)
211        startEnd[1].append(i.endTime)
        arrivalTimes.append(i.arrivalTime)

213
214        if (i.priority == 'LOW'):
215            startEndLow[0].append(i.startTime)
            startEndLow[1].append(i.endTime)
217            arrivalTimesLow.append(i.arrivalTime)

219        if (i.priority == 'HIGH'):
            startEndHigh[0].append(i.startTime)
221            startEndHigh[1].append(i.endTime)
            arrivalTimesHigh.append(i.arrivalTime)

223
224     time = np.arange(0, startEnd[1][-1], 10000)
225     lowPriority = []
        highPriority = []

227
228     for i in range(0, len(time)):
229         length = 0
            for j in range(0, len(arrivalTimesLow)):
231                 if (time[i] >= arrivalTimesLow[j] and startEndLow[1][j] >
time[i]):
                    if (startEndLow[0][j] > arrivalTimesLow[j]):
233                        length += 1
                    lowPriority.append(length)

235
236     for i in range(0, len(time)):
237         length = 0
            for j in range(0, len(arrivalTimesHigh)):
239                 if (time[i] >= arrivalTimesHigh[j] and startEndHigh[1][j] >
time[i]):

```

```

241         if (startEndHigh[0][j] > arrivalTimesHigh[j]):
                length += 1
243         highPriority.append(length)

combined = list( map(add, highPriority, lowPriority) )

245

247     print("Average queue length (LOW) -----", round(sum(
lowPriority)/len(lowPriority), 4), "packets/second")
    print("Average queue length (HIGH) -----", round(sum(
highPriority)/len(highPriority), 4), "packets/second")
249     print("Average queue length (TOTAL) -----", round(sum(combined
)/len(combined), 4), "packets/second")

251

time = time/10**6

253

plt.figure()
255 plt.plot(time, combined, label=str('Combined 1 Mbps'))
plt.legend(loc='best')
257 plt.xlabel('Time (seconds)')
plt.ylabel('Customers waiting in queue')
259 plt.title('Non-preemptive combined queues Trace')
plt.show()

261

plt.figure()
263 plt.plot(time, lowPriority, label=str('Low Priority 1 Mbps'))
plt.plot(time, highPriority, label=str('High Priority 1 Mbps'))
265 plt.legend(loc='best')
plt.xlabel('Time (seconds)')
267 plt.ylabel('Customers waiting in queue')
plt.title('Non-preemptive low and high queues Trace')
269 plt.show()
print(t.time() - timeNow)

271

273 if (doPreemptiveTrace == True):
    print("
#####")
275     print("##### Processing preemptive queuing
#####")
    print("
#####")
277     timeNow = t.time()
    packets = [[],[]]
279     time = []

281     transCap = linkCapacity

283     # Read the csv file data into respective lists to store the data
    interArrivalHigh, packetSizeHigh = readCSV('HighPriority.txt')
285     interArrivalLow, packetSizeLow = readCSV('LowPriority.txt')

287     # Create the list of low priority packets
    for i in range(0, len(interArrivalLow)):
289         lowPacket = Packet(packetSizeLow[i], interArrivalLow[i], 'LOW')

```

```

291         packets[0].append(lowPacket)

293     # Create the list of high priority packets
    for i in range(0, len(interArrivalHigh)):
        highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH'
295     )
        packets[1].append(highPacket)

297     # Get the parameters
    arrivalTimes, transTimes, queuingDelays, startEnd, respTimes, allPackets =
        calcPreemptive(transCap, packets)
299     sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))), float(
        sum(packetSizeHigh))/len(packetSizeHigh)]

301     print("Link capacity -----", transCap/10**6, "
Mbps")
    print("Number of low priority packets -----", len(packetSizeLow),
        "packets")
303     print("Number of high priority packets -----", len(packetSizeHigh)
        , "packets")
    print("Total number of packets -----", len(packetSizeLow)
+ len(packetSizeHigh), "packets" )
305     print("Average inter-arrival time -- (LOW) ----", round((sum(
        interArrivalLow))/(len(interArrivalLow)), 4), "us" )
    print("Average inter-arrival time -- (HIGH) ----", round((sum(
        interArrivalHigh))/(len(interArrivalHigh)), 4), "us" )
307     print("Average inter-arrival time -- (TOTAL) --", round((sum(
        interArrivalHigh) + sum(interArrivalLow))/(len(interArrivalHigh) + len
        (interArrivalLow)), 4), "us" )
    print("Average transmission time --- (LOW) ----", round(sum(
        transTimes[0])/len(transTimes[0]), 4), "us" )
309     print("Average transmission time --- (HIGH) ----", round(sum(
        transTimes[1])/len(transTimes[1]), 4), "us" )
    print("Average transmission time --- (TOTAL) --", round((sum(
        transTimes[1]) + sum(transTimes[0]))/(len(transTimes[1])+len(
        transTimes[0])), 4), "us" )
311     print("Average response time ----- (LOW) ----", round(sum(respTimes
        [0])/len(respTimes[0]), 4), "us" )
    print("Average response time ----- (HIGH) ----", round(sum(respTimes
        [1])/len(respTimes[1]), 4), "us" )
313     print("Average response time ----- (TOTAL) --", round((sum(
        respTimes[0]) + sum(respTimes[1]))/(len(respTimes[0]) + len(respTimes
        [1])), 4), "us" )
    print("The average packet size ----- (LOW) ----", round(sizeAvg[0],
        4), "bits" )
315     print("The average packet size ----- (HIGH) ----", round(sizeAvg[1],
        4), "bits" )
    print("The average packet size ----- (TOTAL) --", round((sum(
        packetSizeLow) + sum(packetSizeHigh)) / (float(len(packetSizeLow)) +
        float(len(packetSizeHigh))), 4), "bits" )
317     print("Average delays ----- (LOW) ----", round(sum(
        queuingDelays[0])/len(queuingDelays[0]), 4), "us")
    print("Average delays ----- (HIGH) ----", round(sum(
        queuingDelays[1])/len(queuingDelays[1]), 4), "us")
319     print("Average delays ----- (TOTAL) --", round((sum(
        queuingDelays[0]) + sum(queuingDelays[1]))/(len(queuingDelays[0]) +

```



```

len(queuingDelays[1])), 4), "us")
    print("Average arrival rate (lambda) (LOW) ----", round((len(
packetSizeLow))/((sum(interArrivalLow))*10**6, 4), "packets/second")
321    print("Average arrival rate (lambda) (HIGH) ---", round((len(
packetSizeHigh))/((sum(interArrivalHigh))*10**6, 4), "packets/second"
)
    print("Average arrival rate (lambda) (TOTAL) --", round((len(
packetSizeLow) + len(packetSizeHigh))/(((sum(interArrivalHigh)*40000 +
sum(interArrivalLow)*50000)/90000))*10**6, 4), "packets/second")
323    print("Average service rate (mu) --- (LOW) ----", round(transCap/
sizeAvg[0], 4), "packets/second")
    print("Average service rate (mu) --- (HIGH) ---", round(transCap/
sizeAvg[1], 4), "packets/second")
325    print("Average service rate (mu) --- (TOTAL) --", round(transCap/(((
sizeAvg[0]*50000 + sizeAvg[1]*40000)/90000)), 4), "packets/second")

327
startEnd = [[],[]]
329 arrivalTimes = []
startEndLow = [[],[]]
331 arrivalTimesLow = []
startEndHigh = [[],[]]
333 arrivalTimesHigh = []

335 for i in allPackets:
    startEnd[0].append(i.startTime)
337     startEnd[1].append(i.endTime)
    arrivalTimes.append(i.arrivalTime)

339
    if (i.priority == 'LOW'):
341         startEndLow[0].append(i.startTime)
        startEndLow[1].append(i.endTime)
343         arrivalTimesLow.append(i.arrivalTime)

345
    if (i.priority == 'HIGH'):
347         startEndHigh[0].append(i.startTime)
        startEndHigh[1].append(i.endTime)
349         arrivalTimesHigh.append(i.arrivalTime)

time = np.arange(0, 120*10**6, 10000)
351 lowPriority = []
highPriority = []
353

355 for i in range(0, len(time)):
    length = 0
357     for j in range(0, len(arrivalTimesLow)):
        if (time[i] >= arrivalTimesLow[j] and startEndLow[0][j] >
time[i] and startEndLow[0][j] > arrivalTimesLow[j]):
359         length += 1
        lowPriority.append(length)

361
363 for i in range(0, len(time)):
    length = 0
    for j in range(0, len(arrivalTimesHigh)):

```

```

365         if (time[i] >= arrivalTimesHigh[j] and startEndHigh[0][j] >
time[i] and startEndHigh[0][j] > arrivalTimesHigh[j]):
366             length += 1
367             highPriority.append(length)

368
369     combined = list( map(add, highPriority, lowPriority) )

370
371     print("Average queue length (LOW) -----", round(sum(
lowPriority)/len(lowPriority), 4), "packets/second")
372     print("Average queue length (HIGH) -----", round(sum(
highPriority)/len(highPriority), 4), "packets/second")
373     print("Average queue length (TOTAL) -----", round(sum(combined
)/len(combined), 4), "packets/second")

374
375     time = time/10**6

376
377     plt.figure()
378     plt.plot(time, combined, label=str('Combined 1 Mbps'))
379     plt.legend(loc='best')
380     plt.xlabel('Time (seconds)')
381     plt.ylabel('Customers waiting in queue')
382     plt.title('Combined Preemptive queues Trace')
383     plt.show()

384
385     plt.figure()
386     plt.plot(time, lowPriority, label=str('Low Priority 1 Mbps'))
387     plt.plot(time, highPriority, label=str('High Priority 1 Mbps'))
388     plt.legend(loc='best')
389     plt.xlabel('Time (seconds)')
390     plt.ylabel('Customers waiting in queue')
391     plt.title('Preemptive queues low and high Trace')
392     plt.show()
393     print(t.time() - timeNow)

```

Code/DriverTrace.py

C Averaging Results Driver

```

# -*- coding: utf-8 -*-
2 """
Created on Mon Sep 16 09:41:05 2019
4
@author: project
6 """
import matplotlib.pyplot as plt
8 import numpy as np
from copy import deepcopy
10 from PriorityQueuing import Packet, calcNoPriority, calcNonPreemptive,
calcPreemptive
from operator import add
12
doAveragePreemptive = False

```

```

14 doAverageNonPreemptive      = False
15 doAverageNoPriority          = True
16
17 averagedOver = 150
18 linkCapacity = 1*10**6
19
20 if (doAverageNonPreemptive == True):
21     print("
22     #####")
23     print("##### Averaged Non-preemptive queuing
24     #####")
25     print("
26     #####")
27
28     time = np.arange(0, 101*10**6, 1000000)
29
30     totalInterArrivalLow      = []
31     totalInterArrivalHigh     = []
32     totalTransmissionLow      = []
33     totalTransmissionHigh     = []
34     totalResponseLow          = []
35     totalResponseHigh         = []
36     totalAveragePacketSizeLow = []
37     totalAveragePacketSizeHigh = []
38     totalAverageDelaysLow      = []
39     totalAverageDelaysHigh     = []
40     totalArrivalRateLow        = []
41     totalArrivalRateHigh       = []
42     totalServiceRateLow        = []
43     totalServiceRateHigh       = []
44     allCombinedLengths         = [[]] * averagedOver
45     allLengthsHighPriority      = [[]] * averagedOver
46     allLengthsLowPriority       = [[]] * averagedOver
47
48     for k in range(0, averagedOver):
49         print(k)
50         packets = [[], []]
51         transCap = linkCapacity
52
53         # Read the csv file data into respective lists to store the data
54         interArrivalHigh = list(np.random.exponential(2486, 40000))
55         packetSizeHigh    = list(np.random.exponential(1000, 40000))
56         interArrivalLow  = list(np.random.exponential(1991, 50000))
57         packetSizeLow    = list(np.random.exponential(1000, 50000))
58
59         # Create the list of low priority packets
60         for i in range(0, len(interArrivalLow)):
61             lowPacket = Packet(packetSizeLow[i], interArrivalLow[i], 'LOW
62             ')
63             packets[0].append(lowPacket)
64
65         # Create the list of high priority packets
66         for i in range(0, len(interArrivalHigh)):
67             highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], '
68             HIGH')

```

```

        packets[1].append(highPacket)

66
    # Get the parameters
68    arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
    allPackets = calcNonPreemptive(transCap, packets)
    sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))),
float(sum(packetSizeHigh))/len(packetSizeHigh)]

70
    if (k == 0):
72        print("Link capacity -----", transCap
/10**6, "Mbps")
        print("Number of low priority packets -----", len(
packetSizeLow), "packets")
74        print("Number of high priority packets -----", len(
packetSizeHigh), "packets")
        print("Total number of packets -----", len(
packetSizeLow) + len(packetSizeHigh), "packets" )

76
        totalInterArrivalLow      += interArrivalLow
78        totalInterArrivalHigh    += interArrivalHigh
        totalTransmissionLow      += transTimes[0]
80        totalTransmissionHigh    += transTimes[1]
        totalResponseLow          += respTimes[0]
82        totalResponseHigh        += respTimes[1]
        totalAveragePacketSizeLow += packetSizeLow
84        totalAveragePacketSizeHigh += packetSizeHigh
        totalAverageDelaysLow     += queuingDelays[0]
86        totalAverageDelaysHigh    += queuingDelays[1]

88        startEnd = [[], []]
        arrivalTimes = []
90        startEndLow = [[], []]
        arrivalTimesLow = []
92        startEndHigh = [[], []]
        arrivalTimesHigh = []

94
    for i in allPackets:
96        startEnd[0].append(i.startTime)
        startEnd[1].append(i.endTime)
98        arrivalTimes.append(i.arrivalTime)

100        if (i.priority == 'LOW'):
            startEndLow[0].append(i.startTime)
102            startEndLow[1].append(i.endTime)
            arrivalTimesLow.append(i.arrivalTime)

104
            if (i.priority == 'HIGH'):
106                startEndHigh[0].append(i.startTime)
                startEndHigh[1].append(i.endTime)
108                arrivalTimesHigh.append(i.arrivalTime)

110        combined = []
        lowPriority = []
112        highPriority = []
114

```

```

116         for i in range(0, len(time)):
117             length = 0
118             for j in range(0, len(arrivalTimesLow)):
119                 if (time[i] >= arrivalTimesLow[j] and startEndLow[1][j] >
time[i]):
120                     if (startEndLow[0][j] > arrivalTimesLow[j]):
121                         length += 1
122                     lowPriority.append(length)
123
124         allLengthsLowPriority[k] = deepcopy(lowPriority)
125
126         for i in range(0, len(time)):
127             length = 0
128             for j in range(0, len(arrivalTimesHigh)):
129                 if (time[i] >= arrivalTimesHigh[j] and startEndHigh[1][j]
> time[i]):
130                     if (startEndHigh[0][j] > arrivalTimesHigh[j]):
131                         length += 1
132                     highPriority.append(length)
133
134         allLengthsHighPriority[k] = deepcopy(highPriority)
135         allCombinedLengths[k] = list( map(add, highPriority, lowPriority)
)
136
137         print("Average inter-arrival time -- (LOW) ----", round((sum(
totalInterArrivalLow))/(len(totalInterArrivalLow)), 4), "us" )
138         print("Average inter-arrival time -- (HIGH) ----", round((sum(
totalInterArrivalHigh))/(len(totalInterArrivalHigh)), 4), "us" )
139         print("Average inter-arrival time -- (TOTAL) --", round((sum(
totalInterArrivalHigh) + sum(totalInterArrivalLow))/(len(
totalInterArrivalHigh) + len(totalInterArrivalLow)), 4), "us" )
140         print("Average transmission time --- (LOW) ----", round(sum(
totalTransmissionLow)/len(totalTransmissionLow), 4), "us" )
141         print("Average transmission time --- (HIGH) ----", round(sum(
totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us" )
142         print("Average transmission time --- (TOTAL) --", round((sum(
totalTransmissionHigh) + sum(totalTransmissionLow))/(len(
totalTransmissionHigh)+len(totalTransmissionLow)), 4), "us" )
143         print("Average response time ----- (LOW) ----", round(sum(
totalResponseLow)/len(totalResponseLow), 4), "us" )
144         print("Average response time ----- (HIGH) ----", round(sum(
totalResponseHigh)/len(totalResponseHigh), 4), "us" )
145         print("Average response time ----- (TOTAL) --", round((sum(
totalResponseLow) + sum(totalResponseHigh))/(len(totalResponseLow) +
len(totalResponseHigh)), 4), "us" )
146         print("The average packet size ----- (LOW) ----", round(sum(
totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow), 4), "bits"
)
147         print("The average packet size ----- (HIGH) ----", round(sum(
totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh), 4), "
bits" )
148         print("The average packet size ----- (TOTAL) --", round((sum(
totalAveragePacketSizeLow) + sum(totalAveragePacketSizeHigh)) / (float
(len(totalAveragePacketSizeLow)) + float(len(
totalAveragePacketSizeHigh))), 4), "bits" )

```

```

148     print("Average delays ----- (LOW) ----", round(sum(
totalAverageDelaysLow)/len(totalAverageDelaysLow), 4), "us")
    print("Average delays ----- (HIGH) ---", round(sum(
totalAverageDelaysHigh)/len(totalAverageDelaysHigh), 4), "us")
150     print("Average delays ----- (TOTAL) --", round((sum(
totalAverageDelaysLow) + sum(totalAverageDelaysHigh))/(len(
totalAverageDelaysHigh) + len(totalAverageDelaysLow)), 4), "us")
    print("Average arrival rate (lambda) (LOW) ----", round((len(
totalAveragePacketSizeLow))/((sum(totalInterArrivalLow))*10**6, 4), "
packets/second")
152     print("Average arrival rate (lambda) (HIGH) ---", round((len(
totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh))*10**6, 4),
"packets/second")
    print("Average arrival rate (lambda) (TOTAL) --", round((len(
totalAveragePacketSizeLow))/((sum(totalInterArrivalLow))*10**6 + (len
(totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh))*10**6, 4)
, "packets/second")
154     print("Average service rate (mu) --- (LOW) ----", round(transCap/(sum
(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)), 4), "
packets/second")
    print("Average service rate (mu) --- (HIGH) ---", round(transCap/(sum
(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)), 4), "
packets/second")
156     print("Average service rate (mu) --- (TOTAL) --", round(transCap/((
sum(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)*50000 +
sum(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)*50000)
/90000), 4), "packets/second")

158
time = time/10**6

160
averagedCombinedLength = [0] * len(allCombinedLengths[0])
162 averagedLowPriorLength = [0] * len(allLengthsLowPriority[0])
averagedHighPrioLength = [0] * len(allLengthsHighPriority[0])
164
for x in range(0, len(allLengthsHighPriority)):
166     for y in range(0, len(allLengthsHighPriority[x])):
        averagedCombinedLength[y] += allCombinedLengths[x][y]
168         averagedLowPriorLength[y] += allLengthsLowPriority[x][y]
        averagedHighPrioLength[y] += allLengthsHighPriority[x][y]
170
averagedCombinedLength = np.asarray(averagedCombinedLength)
172 averagedLowPriorLength = np.asarray(averagedLowPriorLength)
averagedHighPrioLength = np.asarray(averagedHighPrioLength)
174
for x in range(0, len(averagedCombinedLength)):
176     averagedCombinedLength[x] = averagedCombinedLength[x]/
averagedOver
    averagedLowPriorLength[x] = averagedLowPriorLength[x]/
178     averagedOver
    averagedHighPrioLength[x] = averagedHighPrioLength[x]/
averagedOver

180
    print("Average queue length (LOW) -----", round(sum(
averagedLowPriorLength)/len(averagedLowPriorLength), 4), "packets/

```

```

182 second")
    print("Average queue length (HIGH) -----", round(sum(
    averagedHighPrioLength)/len(averagedHighPrioLength), 4), "packets/
    second")
    print("Average queue length (TOTAL) -----", round(sum(
    averagedCombinedLength)/len(averagedCombinedLength), 4), "packets/
    second")

184

186 plt.figure()
    plt.plot(time, averagedCombinedLength, label=str('Combined 1 Mbps'))
188 plt.legend(loc='best')
    plt.xlabel('Time (seconds)')
190 plt.ylabel('Customers waiting in queue')
    plt.title('Averaged combined non-preemptive queues')
192 plt.show()

194 plt.figure()
    plt.plot(time, averagedLowPriorLength, label=str('Low Priority 1 Mbps
    '))
196 plt.plot(time, averagedHighPrioLength, label=str('High Priority 1
    Mbps'))
    plt.legend(loc='best')
198 plt.xlabel('Time (seconds)')
    plt.ylabel('Customers waiting in queue')
200 plt.title('Averaged low and high non-preemptive queues')
    plt.show()
202

204 if (doAveragePreemptive == True):
    print("
    #####")
206     print("##### Averaged Preemptive queuing
    #####")
    print("
    #####")

208

210     time = np.arange(0, 120*10**6, 1000000)

212     totalInterArrivalLow = []
    totalInterArrivalHigh = []
214     totalTransmissionLow = []
    totalTransmissionHigh = []
216     totalResponseLow = []
    totalResponseHigh = []
218     totalAveragePacketSizeLow = []
    totalAveragePacketSizeHigh = []
220     totalAverageDelaysLow = []
    totalAverageDelaysHigh = []
222     totalArrivalRateLow = []
    totalArrivalRateHigh = []
224     totalServiceRateLow = []
    totalServiceRateHigh = []
226     allCombinedLengths = [[]] * averagedOver
    allLengthsHighPriority = [[]] * averagedOver

```

```

228 allLengthsLowPriority      = [[]] * averagedOver
230 for k in range(0, averagedOver):
231     print(k)
232     packets = [], []
233     transCap = linkCapacity
234
235     # Read the csv file data into respective lists to store the data
236     interArrivalHigh = list(np.random.exponential(2486 , 40000))
237     packetSizeHigh   = list(np.random.exponential(1000, 40000))
238     interArrivalLow  = list(np.random.exponential(1991 , 50000))
239     packetSizeLow    = list(np.random.exponential(1000, 50000))
240
241     # Create the list of low priority packets
242     for i in range(0, len(interArrivalLow)):
243         lowPacket = Packet(packetSizeLow[i], interArrivalLow[i], 'LOW
244     ')
245         packets[0].append(lowPacket)
246
247     # Create the list of high priority packets
248     for i in range(0, len(interArrivalHigh)):
249         highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], '
250     HIGH')
251         packets[1].append(highPacket)
252
253     # Get the parameters
254     arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
255     allPackets = calcPreemptive(transCap, packets)
256     sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))),
257     float(sum(packetSizeHigh))/len(packetSizeHigh)]
258
259     if (k == 0):
260         print("Link capacity -----", transCap
261     /10**6, "Mbps")
262         print("Number of low priority packets -----", len(
263     packetSizeLow), "packets")
264         print("Number of high priority packets -----", len(
265     packetSizeHigh), "packets")
266         print("Total number of packets -----", len(
267     packetSizeLow) + len(packetSizeHigh), "packets" )
268
269     totalInterArrivalLow      += interArrivalLow
270     totalInterArrivalHigh     += interArrivalHigh
271     totalTransmissionLow      += transTimes[0]
272     totalTransmissionHigh     += transTimes[1]
273     totalResponseLow          += respTimes[0]
274     totalResponseHigh         += respTimes[1]
275     totalAveragePacketSizeLow += packetSizeLow
276     totalAveragePacketSizeHigh += packetSizeHigh
277     totalAverageDelaysLow     += queuingDelays[0]
278     totalAverageDelaysHigh    += queuingDelays[1]
279
280     startEnd = [], []
281     arrivalTimes = []
282     startEndLow = [], []
283     arrivalTimesLow = []

```



```

276 startEndHigh = [[], []]
    arrivalTimesHigh = []
278
280 for i in allPackets:
    startEnd[0].append(i.startTime)
    startEnd[1].append(i.endTime)
282 arrivalTimes.append(i.arrivalTime)
284
    if (i.priority == 'LOW'):
        startEndLow[0].append(i.startTime)
        startEndLow[1].append(i.endTime)
286 arrivalTimesLow.append(i.arrivalTime)
288
    if (i.priority == 'HIGH'):
        startEndHigh[0].append(i.startTime)
        startEndHigh[1].append(i.endTime)
290 arrivalTimesHigh.append(i.arrivalTime)
292
294 combined = []
    lowPriority = []
296 highPriority = []
298
300 for i in range(0, len(time)):
    length = 0
    for j in range(0, len(arrivalTimesLow)):
302         if (time[i] >= arrivalTimesLow[j] and startEndLow[1][j] >
time[i]):
            if (startEndLow[0][j] > arrivalTimesLow[j]):
304                 length += 1
            lowPriority.append(length)
306
    allLengthsLowPriority[k] = deepcopy(lowPriority)
308
310 for i in range(0, len(time)):
    length = 0
    for j in range(0, len(arrivalTimesHigh)):
312         if (time[i] >= arrivalTimesHigh[j] and startEndHigh[1][j]
> time[i]):
            if (startEndHigh[0][j] > arrivalTimesHigh[j]):
314                 length += 1
            highPriority.append(length)
316
    allLengthsHighPriority[k] = deepcopy(highPriority)
318 allCombinedLengths[k] = list( map(add, highPriority, lowPriority)
)
320
    print("Average inter-arrival time -- (LOW) ----", round((sum(
totalInterArrivalLow))/(len(totalInterArrivalLow)), 4), "us" )
    print("Average inter-arrival time -- (HIGH) ---", round((sum(
totalInterArrivalHigh))/(len(totalInterArrivalHigh)), 4), "us" )
322    print("Average inter-arrival time -- (TOTAL) --", round((sum(
totalInterArrivalHigh) + sum(totalInterArrivalLow))/(len(
totalInterArrivalHigh) + len(totalInterArrivalLow)), 4), "us" )
    print("Average transmission time --- (LOW) ----", round(sum(
totalTransmissionLow)/len(totalTransmissionLow), 4), "us" )

```

```

324     print("Average transmission time --- (HIGH) ---", round(sum(
totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us" )
    print("Average transmission time --- (TOTAL) --", round((sum(
totalTransmissionHigh) + sum(totalTransmissionLow))/(len(
totalTransmissionHigh)+len(totalTransmissionLow)), 4), "us" )
326     print("Average response time ----- (LOW) ---", round(sum(
totalResponseLow)/len(totalResponseLow), 4), "us" )
    print("Average response time ----- (HIGH) ---", round(sum(
totalResponseHigh)/len(totalResponseHigh), 4), "us" )
328     print("Average response time ----- (TOTAL) --", round((sum(
totalResponseLow) + sum(totalResponseHigh))/(len(totalResponseLow) +
len(totalResponseHigh)), 4) , "us" )
    print("The average packet size ----- (LOW) ----", round(sum(
totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow), 4) , "bits"
)
330     print("The average packet size ----- (HIGH) ---", round(sum(
totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh), 4) , "
bits" )
    print("The average packet size ----- (TOTAL) --", round((sum(
totalAveragePacketSizeLow) + sum(totalAveragePacketSizeHigh)) / (float
(len(totalAveragePacketSizeLow)) + float(len(
totalAveragePacketSizeHigh))),4), "bits" )
332     print("Average delays ----- (LOW) ----", round(sum(
totalAverageDelaysLow)/len(totalAverageDelaysLow), 4), "us")
    print("Average delays ----- (HIGH) ---", round(sum(
totalAverageDelaysHigh)/len(totalAverageDelaysHigh), 4), "us")
334     print("Average delays ----- (TOTAL) --", round((sum(
totalAverageDelaysLow) + sum(totalAverageDelaysHigh))/(len(
totalAverageDelaysHigh) + len(totalAverageDelaysLow)), 4), "us")
    print("Average arrival rate (lambda) (LOW) ----", round((len(
totalAveragePacketSizeLow))/((sum(totalInterArrivalLow))*10**6, 4), "
packets/second")
336     print("Average arrival rate (lambda) (HIGH) ---", round((len(
totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh))*10**6, 4),
"packets/second")
    print("Average arrival rate (lambda) (TOTAL) --", round((len(
totalAveragePacketSizeLow))/((sum(totalInterArrivalLow))*10**6 + (len
(totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh))*10**6, 4)
, "packets/second")
338     print("Average service rate (mu) --- (LOW) ----", round(transCap/(sum
(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)), 4), "
packets/second")
    print("Average service rate (mu) --- (HIGH) ---", round(transCap/(sum
(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)), 4), "
packets/second")
340     print("Average service rate (mu) --- (TOTAL) --", round(transCap/((
sum(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)*50000 +
sum(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)*50000)
/90000), 4), "packets/second")
    time = time/10**6

342     averagedCombinedLength = [0] * len(allCombinedLengths[0])
344     averagedLowPriorLength = [0] * len(allLengthsLowPriority[0])
    averagedHighPrioLength = [0] * len(allLengthsHighPriority[0])

346     for x in range(0, len(allLengthsHighPriority)):

```

```

348     for y in range(0, len(allLengthsHighPriority[x])):
350         averagedCombinedLength[y] += allCombinedLengths[x][y]
352         averagedLowPriorLength[y] += allLengthsLowPriority[x][y]
354         averagedHighPrioLength[y] += allLengthsHighPriority[x][y]
356
358     averagedCombinedLength = np.asarray(averagedCombinedLength)
360     averagedLowPriorLength = np.asarray(averagedLowPriorLength)
362     averagedHighPrioLength = np.asarray(averagedHighPrioLength)
364
366     for x in range(0, len(averagedCombinedLength)):
368         averagedCombinedLength[x] = averagedCombinedLength[x]/
370         averagedOver
372         averagedLowPriorLength[x] = averagedLowPriorLength[x]/
374         averagedOver
376         averagedHighPrioLength[x] = averagedHighPrioLength[x]/
378         averagedOver
380
382     print("Average queue length (LOW) -----", round(sum(
384         averagedLowPriorLength)/len(averagedLowPriorLength), 4), "packets/
386         second")
388     print("Average queue length (HIGH) -----", round(sum(
390         averagedHighPrioLength)/len(averagedHighPrioLength), 4), "packets/
392         second")
394     print("Average queue length (TOTAL) -----", round(sum(
396         averagedCombinedLength)/len(averagedCombinedLength), 4), "packets/
398         second")
400
402     plt.figure()
404     plt.plot(time, averagedCombinedLength, label=str('Combined 1 Mbps'))
406     plt.legend(loc='best')
408     plt.xlabel('Time (seconds)')
410     plt.ylabel('Customers waiting in queue')
412     plt.title('Averaged combined preemptive queues')
414     plt.show()
416
418     plt.figure()
420     plt.plot(time, averagedLowPriorLength, label=str('Low Priority 1 Mbps
422     '))
424     plt.plot(time, averagedHighPrioLength, label=str('High Priority 1
426     Mbps'))
428     plt.legend(loc='best')
430     plt.xlabel('Time (seconds)')
432     plt.ylabel('Customers waiting in queue')
434     plt.title('Averaged low and high preemptive queues')
436     plt.show()
438
440 if (doAverageNoPriority == True):
442     print("
444     #####")
446     print("##### Averaged No Priority queuing
448     #####")
450     print("
452     #####")

```

```

390 time = np.arange(0, 101*10**6, 1000000)
392
394 totalInterArrivalLow      = []
394 totalInterArrivalHigh    = []
396 totalTransmissionLow     = []
396 totalTransmissionHigh   = []
398 totalResponseLow         = []
398 totalResponseHigh       = []
400 totalAveragePacketSizeLow = []
400 totalAveragePacketSizeHigh = []
402 totalAverageDelaysLow    = []
402 totalAverageDelaysHigh   = []
404 totalArrivalRateLow      = []
404 totalArrivalRateHigh     = []
406 totalServiceRateLow     = []
406 totalServiceRateHigh    = []
408 allCombinedLengths       = [[]] * averagedOver
408 allLengthsHighPriority    = [[]] * averagedOver
410 allLengthsLowPriority     = [[]] * averagedOver
412
412 for k in range(0, averagedOver):
412     print(k)
414     packets = [], []
414     transCap = linkCapacity
416
416     # Read the csv file data into respective lists to store the data
418     # Read the csv file data into respective lists to store the data
418     interArrivalHigh = list(np.random.exponential(2486 , 40000))
420     packetSizeHigh   = list(np.random.exponential(1000, 40000))
420     interArrivalLow  = list(np.random.exponential(1991 , 50000))
422     packetSizeLow    = list(np.random.exponential(1000, 50000))
424
424     # Create the list of low priority packets
424     for i in range(0, len(interArrivalLow)):
426         lowPacket = Packet(packetSizeLow[i], interArrivalLow[i], 'LOW
426         ')
426         packets[0].append(lowPacket)
428
428     # Create the list of high priority packets
430     for i in range(0, len(interArrivalHigh)):
430         highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], '
430         HIGH')
432         packets[1].append(highPacket)
434
434     # Get the parameters
434     arrivalTimes , transTimes , queuingDelays , startEnd , respTimes ,
434     allPackets = calcNoPriority(transCap, packets)
436     sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))),
436     float(sum(packetSizeHigh))/len(packetSizeHigh)]
438
438     if (k == 0):
438         print("Link capacity -----", transCap
438         /10**6, "Mbps")

```

```

440         print("Number of low priority packets -----", len(
packetSizeLow), "packets")
         print("Number of high priority packets -----", len(
packetSizeHigh), "packets")
442         print("Total number of packets -----", len(
packetSizeLow) + len(packetSizeHigh), "packets" )

444         totalInterArrivalLow      += interArrivalLow
         totalInterArrivalHigh      += interArrivalHigh
446         totalTransmissionLow      += transTimes[0]
         totalTransmissionHigh      += transTimes[1]
448         totalResponseLow          += respTimes[0]
         totalResponseHigh          += respTimes[1]
450         totalAveragePacketSizeLow += packetSizeLow
         totalAveragePacketSizeHigh += packetSizeHigh
452         totalAverageDelaysLow      += queuingDelays[0]
         totalAverageDelaysHigh      += queuingDelays[1]
454
         startEnd = [[],[]]
456         arrivalTimes = []
         startEndLow = [[],[]]
458         arrivalTimesLow = []
         startEndHigh = [[],[]]
460         arrivalTimesHigh = []

462         for i in allPackets:
         startEnd[0].append(i.startTime)
464         startEnd[1].append(i.endTime)
         arrivalTimes.append(i.arrivalTime)

466
         if (i.priority == 'LOW'):
468             startEndLow[0].append(i.startTime)
             startEndLow[1].append(i.endTime)
470             arrivalTimesLow.append(i.arrivalTime)

472             if (i.priority == 'HIGH'):
             startEndHigh[0].append(i.startTime)
474             startEndHigh[1].append(i.endTime)
             arrivalTimesHigh.append(i.arrivalTime)
476

         combined = []
478         lowPriority = []
         highPriority = []
480
         for i in range(0, len(time)):
482             length = 0
             for j in range(0, len(arrivalTimesLow)):
484                 if (time[i] >= arrivalTimesLow[j] and startEndLow[1][j] >
time[i]):
                     if (startEndLow[0][j] > arrivalTimesLow[j]):
486                         length += 1
                     lowPriority.append(length)
488
         allLengthsLowPriority[k] = deepcopy(lowPriority)
490
         for i in range(0, len(time)):

```

```

492         length = 0
494         for j in range(0, len(arrivalTimesHigh)):
496             if (time[i] >= arrivalTimesHigh[j] and startEndHigh[1][j]
> time[i]):
498                 if (startEndHigh[0][j] > arrivalTimesHigh[j]):
500                     length += 1
502                     highPriority.append(length)
504
506         allLengthsHighPriority[k] = deepcopy(highPriority)
508         allCombinedLengths[k] = list( map(add, highPriority, lowPriority)
)

510     print("Average inter-arrival time -- (LOW) ----", round((sum(
totalInterArrivalLow))/(len(totalInterArrivalLow)), 4), "us" )
512     print("Average inter-arrival time -- (HIGH) ---", round((sum(
totalInterArrivalHigh))/(len(totalInterArrivalHigh)), 4), "us" )
514     print("Average inter-arrival time -- (TOTAL) --", round((sum(
totalInterArrivalHigh) + sum(totalInterArrivalLow))/(len(
totalInterArrivalHigh) + len(totalInterArrivalLow)), 4), "us" )
516     print("Average transmission time --- (LOW) ----", round(sum(
totalTransmissionLow)/len(totalTransmissionLow), 4), "us" )
518     print("Average transmission time --- (HIGH) ---", round(sum(
totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us" )
520     print("Average transmission time --- (TOTAL) --", round((sum(
totalTransmissionHigh) + sum(totalTransmissionLow))/(len(
totalTransmissionHigh)+len(totalTransmissionLow)), 4), "us" )
522     print("Average response time ----- (LOW) ---", round(sum(
totalResponseLow)/len(totalResponseLow), 4), "us" )
524     print("Average response time ----- (HIGH) ---", round(sum(
totalResponseHigh)/len(totalResponseHigh), 4), "us" )
526     print("Average response time ----- (TOTAL) --", round((sum(
totalResponseLow) + sum(totalResponseHigh))/(len(totalResponseLow) +
len(totalResponseHigh)), 4), "us" )
528     print("The average packet size ----- (LOW) ----", round(sum(
totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow), 4), "bits"
)
530     print("The average packet size ----- (HIGH) ---", round(sum(
totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh), 4), "
bits" )
532     print("The average packet size ----- (TOTAL) --", round((sum(
totalAveragePacketSizeLow) + sum(totalAveragePacketSizeHigh)) / (float(
len(totalAveragePacketSizeLow)) + float(len(
totalAveragePacketSizeHigh))), 4), "bits" )
534     print("Average delays ----- (LOW) ----", round(sum(
totalAverageDelaysLow)/len(totalAverageDelaysLow), 4), "us" )
536     print("Average delays ----- (HIGH) ---", round(sum(
totalAverageDelaysHigh)/len(totalAverageDelaysHigh), 4), "us" )
538     print("Average delays ----- (TOTAL) --", round((sum(
totalAverageDelaysLow) + sum(totalAverageDelaysHigh))/(len(
totalAverageDelaysHigh) + len(totalAverageDelaysLow)), 4), "us" )
540     print("Average arrival rate (lambda) (LOW) ----", round((len(
totalAveragePacketSizeLow))/((sum(totalInterArrivalLow))*10**6, 4), "
packets/second" )
542     print("Average arrival rate (lambda) (HIGH) ---", round((len(
totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh))*10**6, 4),
"packets/second" )

```

```

print("Average arrival rate (lambda) (TOTAL) --", round((len(
totalAveragePacketSizeLow))/((sum(totalInterArrivalLow))*10**6 + (len
(totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh))*10**6, 4)
, "packets/second")
520 print("Average service rate (mu) --- (LOW) ----", round(transCap/(sum
(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)), 4), "
packets/second")
print("Average service rate (mu) --- (HIGH) ----", round(transCap/(sum
(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)), 4), "
packets/second")
522 print("Average service rate (mu) --- (TOTAL) --", round(transCap/((
sum(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)*50000 +
sum(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)*50000)
/90000), 4), "packets/second")
time = time/10**6

524
526 averagedCombinedLength = [0] * len(allCombinedLengths[0])
averagedLowPriorLength = [0] * len(allLengthsLowPriority[0])
averagedHighPrioLength = [0] * len(allLengthsHighPriority[0])

528
for x in range(0, len(allLengthsHighPriority)):
530     for y in range(0, len(allLengthsHighPriority[x])):
        averagedCombinedLength[y] += allCombinedLengths[x][y]
532         averagedLowPriorLength[y] += allLengthsLowPriority[x][y]
        averagedHighPrioLength[y] += allLengthsHighPriority[x][y]
534
averagedCombinedLength = np.asarray(averagedCombinedLength)
536 averagedLowPriorLength = np.asarray(averagedLowPriorLength)
averagedHighPrioLength = np.asarray(averagedHighPrioLength)
538

540 for x in range(0, len(averagedCombinedLength)):
    averagedCombinedLength[x] = averagedCombinedLength[x]/
averagedOver
542     averagedLowPriorLength[x] = averagedLowPriorLength[x]/
averagedOver
    averagedHighPrioLength[x] = averagedHighPrioLength[x]/
averagedOver
544

print("Average queue length (LOW) -----", round(sum(
averagedLowPriorLength)/len(averagedLowPriorLength), 4), "packets/
second")
546 print("Average queue length (HIGH) -----", round(sum(
averagedHighPrioLength)/len(averagedHighPrioLength), 4), "packets/
second")
print("Average queue length (TOTAL) -----", round(sum(
averagedCombinedLength)/len(averagedCombinedLength), 4), "packets/
second")
548

550 plt.figure()
plt.plot(time, averagedCombinedLength, label=str('Combined 1 Mbps'))
552 plt.legend(loc='best')
plt.xlabel('Time (seconds)')
554 plt.ylabel('Customers waiting in queue')
plt.title('Averaged combined no-priority queues')

```

```

556     plt.show()

558     plt.figure()
    plt.plot(time, averagedLowPriorLength, label=str('Low Priority 1 Mbps'))
560     plt.plot(time, averagedHighPrioLength, label=str('High Priority 1 Mbps'))
    plt.legend(loc='best')
562     plt.xlabel('Time (seconds)')
    plt.ylabel('Customers waiting in queue')
564     plt.title('Averaged low and high no-priority queues')
    plt.show()

```

Code/DriverAverage.py

D Averaging Varying Arrival Rates Driver

```

1  # -*- coding: utf-8 -*-
   """
3  Created on Mon Sep 16 09:41:12 2019

5  @author: project
   """
7  import matplotlib.pyplot as plt
   from PriorityQueuing import Packet, calcNoPriority, calcNonPreemptive,
       calcPreemptive
9  import numpy as np
   from operator import add
11 from copy import deepcopy

13 doVaryingArrivalPreempt      = False
   doVaryingArrivalNonPreempt  = True
15 doVaryingArrivalNoPriority   = False

17 averagedOver = 5
   print("Averaged over:", averagedOver)
19 arrRates = [2000] # Actually interArrivalTimes
   linkCapacity = 1*10**6
21

23 if (doVaryingArrivalNonPreempt == True):
    print("
#####")
25     print("##### Averaged Non-Preemptive varying arrRate
#####")
    print("
#####")

27     allQueuesLow  = []
29     allQueuesHigh = []
    allQueuesComb  = []

31     for l in range(0, len(arrRates)):

```



```

33     time = np.arange(0, 101*10**6, 1000000)

35
37     totalInterArrivalLow      = []
38     totalInterArrivalHigh     = []
39     totalTransmissionLow      = []
40     totalTransmissionHigh     = []
41     totalResponseLow          = []
42     totalResponseHigh         = []
43     totalAveragePacketSizeLow = []
44     totalAveragePacketSizeHigh = []
45     totalAverageDelaysLow     = []
46     totalAverageDelaysHigh    = []
47     totalArrivalRateLow       = []
48     totalArrivalRateHigh      = []
49     totalServiceRateLow       = []
50     totalServiceRateHigh      = []
51     allCombinedLengths        = [[]] * averagedOver
52     allLengthsHighPriority     = [[]] * averagedOver
53     allLengthsLowPriority      = [[]] * averagedOver
54
55     for k in range(0, averagedOver):
56         packets = [[] , []]
57         transCap = linkCapacity
58         print(l,k)
59         # Read the csv file data into respective lists to store the
data
60         interArrivalHigh = list(np.random.exponential(arrRates[l] ,
int(100/(arrRates[l]*10**-6))))
61         packetSizeHigh    = list(np.random.exponential(1000, int(100/(
arrRates[l]*10**-6))))
62         interArrivalLow  = list(np.random.exponential(2000 , 50000))
63         packetSizeLow    = list(np.random.exponential(1000, 50000))
64
65         # Create the list of low priority packets
66         for i in range(0, len(interArrivalLow)):
LOW')
67             lowPacket = Packet(packetSizeLow[i], interArrivalLow[i],
68
69             packets[0].append(lowPacket)
70
71             # Create the list of high priority packets
72             for i in range(0, len(interArrivalHigh)):
73                 highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i]
74
75                 packets[1].append(highPacket)
76
77             # Get the parameters
78             arrivalTimes , transTimes , queuingDelays , startEnd , respTimes ,
allPackets = calcNonPreemptive(transCap , packets)
79             sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))),
float(sum(packetSizeHigh))/len(packetSizeHigh)]
80             if (k == 0):
81                 print("Link capacity -----",
transCap/10**6, "Mbps")
82                 print("Number of low priority packets -----", len(
packetSizeLow), "packets")

```

```

81         print("Number of high priority packets -----", len(
            packetSizeHigh), "packets")
        print("Total number of packets -----", len(
            packetSizeLow) + len(packetSizeHigh), "packets" )

83         totalInterArrivalLow      += interArrivalLow
            totalInterArrivalHigh    += interArrivalHigh
85         totalTransmissionLow      += transTimes[0]
            totalTransmissionHigh    += transTimes[1]
87         totalResponseLow          += respTimes[0]
            totalResponseHigh        += respTimes[1]
89         totalAveragePacketSizeLow += packetSizeLow
            totalAveragePacketSizeHigh += packetSizeHigh
91         totalAverageDelaysLow     += queuingDelays[0]
            totalAverageDelaysHigh    += queuingDelays[1]
93
            startEnd = [[], []]
            arrivalTimes = []
95         startEndLow = [[], []]
            arrivalTimesLow = []
97         startEndHigh = [[], []]
            arrivalTimesHigh = []
99
101        for i in allPackets:
            startEnd[0].append(i.startTime)
103            startEnd[1].append(i.endTime)
            arrivalTimes.append(i.arrivalTime)
105
            if (i.priority == 'LOW'):
107                startEndLow[0].append(i.startTime)
                    startEndLow[1].append(i.endTime)
109                arrivalTimesLow.append(i.arrivalTime)
111
            if (i.priority == 'HIGH'):
113                startEndHigh[0].append(i.startTime)
                    startEndHigh[1].append(i.endTime)
115                arrivalTimesHigh.append(i.arrivalTime)
117
            lowPriority = []
            highPriority = []
119
121        for i in range(0, len(time)):
            length = 0
            for j in range(0, len(arrivalTimesLow)):
123                if (time[i] >= arrivalTimesLow[j] and startEndLow[1][
                    j] > time[i]):
125                    if (startEndLow[0][j] > arrivalTimesLow[j]):
                        length += 1
                    lowPriority.append(length)
127
            allLengthsLowPriority[k] = deepcopy(lowPriority)
129
131        for i in range(0, len(time)):
            length = 0
            for j in range(0, len(arrivalTimesHigh)):

```

```

133         if (time[i] >= arrivalTimesHigh[j] and startEndHigh
[1][j] > time[i]):
135             if (startEndHigh[0][j] > arrivalTimesHigh[j]):
length += 1
highPriority.append(length)
137
allLengthsHighPriority[k] = deepcopy(highPriority)
139
allCombinedLengths[k] = list( map(add, highPriority,
lowPriority) )
141
print("Average inter-arrival time -- (LOW) ----", round((sum(
totalInterArrivalLow))/(len(totalInterArrivalLow)), 4), "us" )
143 print("Average inter-arrival time -- (HIGH) ---", round((sum(
totalInterArrivalHigh))/(len(totalInterArrivalHigh)), 4), "us" )
print("Average inter-arrival time -- (TOTAL) --", round((sum(
totalInterArrivalHigh) + sum(totalInterArrivalLow))/(len(
totalInterArrivalHigh) + len(totalInterArrivalLow)), 4), "us" )
145 print("Average transmission time --- (LOW) ----", round(sum(
totalTransmissionLow)/len(totalTransmissionLow), 4), "us" )
print("Average transmission time --- (HIGH) ---", round(sum(
totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us" )
147 print("Average transmission time --- (TOTAL) --", round((sum(
totalTransmissionHigh) + sum(totalTransmissionLow))/(len(
totalTransmissionHigh)+len(totalTransmissionLow)), 4), "us" )
print("Average response time ----- (LOW) ---", round(sum(
totalResponseLow)/len(totalResponseLow), 4), "us" )
149 print("Average response time ----- (HIGH) ---", round(sum(
totalResponseHigh)/len(totalResponseHigh), 4), "us" )
print("Average response time ----- (TOTAL) --", round((sum(
totalResponseLow) + sum(totalResponseHigh))/(len(totalResponseLow) +
len(totalResponseHigh)), 4), "us" )
151 print("The average packet size ----- (LOW) ----", round(sum(
totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow), 4), "bits"
)
print("The average packet size ----- (HIGH) ---", round(sum(
totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh), 4), "
bits" )
153 print("The average packet size ----- (TOTAL) --", round((sum(
totalAveragePacketSizeLow) + sum(totalAveragePacketSizeHigh)) / (float
(len(totalAveragePacketSizeLow)) + float(len(
totalAveragePacketSizeHigh))), 4), "bits" )
print("Average delays ----- (LOW) ----", round(sum(
totalAverageDelaysLow)/len(totalAverageDelaysLow), 4), "us")
155 print("Average delays ----- (HIGH) ---", round(sum(
totalAverageDelaysHigh)/len(totalAverageDelaysHigh), 4), "us")
print("Average delays ----- (TOTAL) --", round((sum(
totalAverageDelaysLow) + sum(totalAverageDelaysHigh))/(len(
totalAverageDelaysHigh) + len(totalAverageDelaysLow)), 4), "us")
157 print("Average arrival rate (lambda) (LOW) ----", round((len(
totalAveragePacketSizeLow))/((sum(totalInterArrivalLow))*10**6, 4), "
packets/second")
print("Average arrival rate (lambda) (HIGH) ---", round((len(
totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh))*10**6, 4),
"packets/second")

```

```

159     print("Average arrival rate (lambda) (TOTAL) --", round((len(
totalAveragePacketSizeLow) + len(totalAveragePacketSizeHigh))/((sum(
totalInterArrivalHigh) + sum(totalInterArrivalLow))*10**6, 4), "
packets/second")
    print("Average service rate (mu) --- (LOW) ----", round(transCap
/(sum(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)), 4),
"packets/second")
161     print("Average service rate (mu) --- (HIGH) ----", round(transCap
/(sum(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)), 4)
, "packets/second")
    print("Average service rate (mu) --- (TOTAL) --", round(transCap
/((sum(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)*50000
+ sum(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)*
arrRates[1]*10**-6)/(50000+arrRates[1]*10**-6)), 4), "packets/second")
163
165     time = time/10**6
167
169     averagedCombinedLength = [0] * len(allCombinedLengths[0])
    averagedLowPriorLength = [0] * len(allLengthsLowPriority[0])
    averagedHighPrioLength = [0] * len(allLengthsHighPriority[0])
171
    for x in range(0, len(allLengthsHighPriority)):
        for y in range(0, len(allLengthsHighPriority[x])):
173            averagedCombinedLength[y] += allCombinedLengths[x][y]
            averagedLowPriorLength[y] += allLengthsLowPriority[x][y]
            averagedHighPrioLength[y] += allLengthsHighPriority[x][y]
175
177     averagedCombinedLength = np.asarray(averagedCombinedLength)
    averagedLowPriorLength = np.asarray(averagedLowPriorLength)
    averagedHighPrioLength = np.asarray(averagedHighPrioLength)
179
181     for x in range(0, len(averagedCombinedLength)):
        averagedCombinedLength[x] = averagedCombinedLength[x]/
averagedOver
183         averagedLowPriorLength[x] = averagedLowPriorLength[x]/
averagedOver
        averagedHighPrioLength[x] = averagedHighPrioLength[x]/
averagedOver
185
187     allQueuesComb.append(deepcopy(averagedCombinedLength))
    allQueuesHigh.append(deepcopy(averagedHighPrioLength))
    allQueuesLow.append(deepcopy(averagedLowPriorLength))
189
191     for a in range(0, len(allQueuesLow)):
        print("Average queue length low priority " + str(arrRates[a]),
round(sum(allQueuesLow[a])/len(allQueuesLow[a]), 4), "packets/second")
        print("Average queue length high priority " + str(arrRates[a])
, round(sum(allQueuesHigh[a])/len(allQueuesHigh[a]), 4), "packets/
second")
193
195     plt.figure()
    for t in range(0, len(allQueuesHigh)):
        plt.plot(time, allQueuesLow[t], label="Low " + str(arrRates[t]) +
" us")
197

```

```

199 plt.legend(loc='best')
plt.xlabel('Time (seconds)')
plt.ylabel('Customers waiting in queue')
201 plt.title('Non-Preemptive Varying arrRate Low LowPriority')
plt.show()

203
plt.figure()
205 for t in range(0, len(allQueuesHigh)):
    plt.plot(time, allQueuesHigh[t], label="High " + str(arrRates[t])
+ " us")
207 plt.legend(loc='best')
plt.xlabel('Time (seconds)')
209 plt.ylabel('Customers waiting in queue')
plt.title('Non-Preemptive Varying arrRate High Priority')
211 plt.show()

213 if( doVaryingArrivalPreempt == True):
    print("
#####")
215    print("##### Averaged Preemptive varying arrRate
#####")
    print("
#####")

217
    allQueuesLow = []
219    allQueuesHigh = []
    allQueuesComb = []

221
    for l in range(0, len(arrRates)):
223        time = np.arange(0, 125*10**6, 1000000)

225
        totalInterArrivalLow = []
227        totalInterArrivalHigh = []
        totalTransmissionLow = []
229        totalTransmissionHigh = []
        totalResponseLow = []
231        totalResponseHigh = []
        totalAveragePacketSizeLow = []
233        totalAveragePacketSizeHigh = []
        totalAverageDelaysLow = []
235        totalAverageDelaysHigh = []
        totalArrivalRateLow = []
237        totalArrivalRateHigh = []
        totalServiceRateLow = []
239        totalServiceRateHigh = []
        allCombinedLengths = [[] * averagedOver
241        allLengthsHighPriority = [[] * averagedOver
        allLengthsLowPriority = [[] * averagedOver

243
    for k in range(0, averagedOver):
245        packets = [[], []]
        transCap = linkCapacity

247
        # Read the csv file data into respective lists to store the
data

```

```

249         interArrivalHigh = list(np.random.exponential(arrRates[1] ,
int(100/(arrRates[1]*10**-6))))
        packetSizeHigh = list(np.random.exponential(1000, int(100/(
arrRates[1]*10**-6))))
251         interArrivalLow = list(np.random.exponential(2000 , 50000))
        packetSizeLow = list(np.random.exponential(1000, 50000))
253
        # Create the list of low priority packets
255         for i in range(0, len(interArrivalLow)):
            lowPacket = Packet(packetSizeLow[i], interArrivalLow[i],
'LOW')
257             packets[0].append(lowPacket)

        # Create the list of high priority packets
259         for i in range(0, len(interArrivalHigh)):
            highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i]
], 'HIGH')
261             packets[1].append(highPacket)

263
        # Get the parameters
265         arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
allPackets = calcPreemptive(transCap, packets)
        sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))),
float(sum(packetSizeHigh))/len(packetSizeHigh)]
267

        if (k == 0):
269             print("Link capacity -----",
transCap/10**6, "Mbps")
            print("Number of low priority packets -----", len(
packetSizeLow), "packets")
271             print("Number of high priority packets -----", len(
packetSizeHigh), "packets")
            print("Total number of packets -----", len(
packetSizeLow) + len(packetSizeHigh), "packets" )

273
        print(l,k)
275         totalInterArrivalLow += interArrivalLow
        totalInterArrivalHigh += interArrivalHigh
277         totalTransmissionLow += transTimes[0]
        totalTransmissionHigh += transTimes[1]
279         totalResponseLow += respTimes[0]
        totalResponseHigh += respTimes[1]
281         totalAveragePacketSizeLow += packetSizeLow
        totalAveragePacketSizeHigh += packetSizeHigh
283         totalAverageDelaysLow += queuingDelays[0]
        totalAverageDelaysHigh += queuingDelays[1]
285

        startEnd = [[],[]]
287         arrivalTimes = []
        startEndLow = [[],[]]
289         arrivalTimesLow = []
        startEndHigh = [[],[]]
291         arrivalTimesHigh = []

293         for i in allPackets:
            startEnd[0].append(i.startTime)

```

```

295         startEnd[1].append(i.endTime)
        arrivalTimes.append(i.arrivalTime)

297
299         if (i.priority == 'LOW'):
301             startEndLow[0].append(i.startTime)
303             startEndLow[1].append(i.endTime)
305             arrivalTimesLow.append(i.arrivalTime)

307
309             if (i.priority == 'HIGH'):
311                 startEndHigh[0].append(i.startTime)
313                 startEndHigh[1].append(i.endTime)
315                 arrivalTimesHigh.append(i.arrivalTime)

317
319         combined = []
321         lowPriority = []
323         highPriority = []

325         for i in range(0, len(time)):
327             length = 0
329             for j in range(0, len(arrivalTimesLow)):
331                 if (time[i] >= arrivalTimesLow[j] and startEndLow[1][
j] > time[i]):
333                     if (startEndLow[0][j] > arrivalTimesLow[j]):
335                         length += 1
337                     lowPriority.append(length)

339         allLengthsLowPriority[k] = deepcopy(lowPriority)

341         for i in range(0, len(time)):
343             length = 0
345             for j in range(0, len(arrivalTimesHigh)):
347                 if (time[i] >= arrivalTimesHigh[j] and startEndHigh
[1][j] > time[i]):
349                     if (startEndHigh[0][j] > arrivalTimesHigh[j]):
351                         length += 1
353                     highPriority.append(length)

355         allLengthsHighPriority[k] = deepcopy(highPriority)
357         allCombinedLengths[k] = list( map(add, highPriority,
lowPriority) )

359
361         print("Average inter-arrival time -- (LOW) ----", round((sum(
totalInterArrivalLow))/(len(totalInterArrivalLow)), 4), "us" )
363         print("Average inter-arrival time -- (HIGH) ----", round((sum(
totalInterArrivalHigh))/(len(totalInterArrivalHigh)), 4), "us" )
365         print("Average inter-arrival time -- (TOTAL) --", round((sum(
totalInterArrivalHigh) + sum(totalInterArrivalLow))/(len(
totalInterArrivalHigh) + len(totalInterArrivalLow)), 4), "us" )
367         print("Average transmission time --- (LOW) ----", round(sum(
totalTransmissionLow)/len(totalTransmissionLow), 4), "us" )
369         print("Average transmission time --- (HIGH) ----", round(sum(
totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us" )
371         print("Average transmission time --- (TOTAL) --", round((sum(
totalTransmissionHigh) + sum(totalTransmissionLow))/(len(
totalTransmissionHigh)+len(totalTransmissionLow)), 4), "us" )

```

```

341     print("Average response time ----- (LOW) ----", round(sum(
totalResponseLow)/len(totalResponseLow), 4), "us" )
343     print("Average response time ----- (HIGH) ----", round(sum(
totalResponseHigh)/len(totalResponseHigh), 4), "us" )
    print("Average response time ----- (TOTAL) --", round((sum(
totalResponseLow) + sum(totalResponseHigh))/(len(totalResponseLow) +
len(totalResponseHigh)), 4), "us" )
345     print("The average packet size ----- (LOW) ----", round(sum(
totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow), 4), "bits"
)
    print("The average packet size ----- (HIGH) ----", round(sum(
totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh), 4), "
bits" )
    print("The average packet size ----- (TOTAL) --", round((sum(
totalAveragePacketSizeLow) + sum(totalAveragePacketSizeHigh)) / (float
(len(totalAveragePacketSizeLow)) + float(len(
totalAveragePacketSizeHigh))),4), "bits" )
    print("Average delays ----- (LOW) ----", round(sum(
totalAverageDelaysLow)/len(totalAverageDelaysLow), 4), "us")
347     print("Average delays ----- (HIGH) ----", round(sum(
totalAverageDelaysHigh)/len(totalAverageDelaysHigh), 4), "us")
    print("Average delays ----- (TOTAL) --", round((sum(
totalAverageDelaysLow) + sum(totalAverageDelaysHigh))/(len(
totalAverageDelaysHigh) + len(totalAverageDelaysLow)), 4), "us")
349     print("Average arrival rate (lambda) (LOW) ----", round((len(
totalAveragePacketSizeLow))/((sum(totalInterArrivalLow))*10**6, 4), "
packets/second")
    print("Average arrival rate (lambda) (HIGH) ----", round((len(
totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh))*10**6, 4),
"packets/second")
351     print("Average arrival rate (lambda) (TOTAL) --", round((len(
totalAveragePacketSizeLow) + len(totalAveragePacketSizeHigh))/((sum(
totalInterArrivalHigh) + sum(totalInterArrivalLow))*10**6, 4), "
packets/second")
    print("Average service rate (mu) --- (LOW) ----", round(transCap
/(sum(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)), 4),
"packets/second")
353     print("Average service rate (mu) --- (HIGH) ----", round(transCap
/(sum(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)), 4)
, "packets/second")
    print("Average service rate (mu) --- (TOTAL) --", round(transCap
/((sum(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)*50000
+ sum(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)*
arrRates[1]*10**-6)/(50000+arrRates[1]*10**-6)), 4), "packets/second")
355
    time = time/10**6
357
    averagedCombinedLength = [0] * len(allCombinedLengths[0])
359    averagedLowPriorLength = [0] * len(allLengthsLowPriority[0])
    averagedHighPrioLength = [0] * len(allLengthsHighPriority[0])
361
    for x in range(0, len(allLengthsHighPriority)):
363        for y in range(0, len(allLengthsHighPriority[x])):
            averagedCombinedLength[y] += allCombinedLengths[x][y]
365            averagedLowPriorLength[y] += allLengthsLowPriority[x][y]
            averagedHighPrioLength[y] += allLengthsHighPriority[x][y]

```



```

367     averagedCombinedLength = np.asarray(averagedCombinedLength)
369     averagedLowPriorLength = np.asarray(averagedLowPriorLength)
371     averagedHighPrioLength = np.asarray(averagedHighPrioLength)

373     for x in range(0, len(averagedCombinedLength)):
375         averagedCombinedLength[x] = averagedCombinedLength[x]/
averagedOver
377         averagedLowPriorLength[x] = averagedLowPriorLength[x]/
averagedOver
379         averagedHighPrioLength[x] = averagedHighPrioLength[x]/
averagedOver

381     allQueuesComb.append(deepcopy(averagedCombinedLength))
383     allQueuesHigh.append(deepcopy(averagedHighPrioLength))
385     allQueuesLow.append(deepcopy(averagedLowPriorLength))

387     averagedQueuesHigh = [0]*len(allQueuesHigh[0])

389     for a in range(0, len(allQueuesLow)):
391         print("Average queue length low priority " + str(arrRates[a]),
round(sum(allQueuesLow[a])/len(allQueuesLow[a]), 4), "packets/second")
393         print("Average queue length high priority " + str(arrRates[a])
, round(sum(allQueuesHigh[a])/len(allQueuesHigh[a]), 4), "packets/
second")

395     plt.figure()
397     for t in range(0, len(allQueuesHigh)):
399         plt.plot(time, allQueuesLow[t], label="Low " + str(arrRates[t]) +
" us")

401     plt.legend(loc='best')
403     plt.xlabel('Time (seconds)')
405     plt.ylabel('Customers waiting in queue')
407     plt.title('Preemptive Varying arrRate Low LowPriority')
409     plt.show()

411     plt.figure()
413     for t in range(0, len(allQueuesHigh)):
415         plt.plot(time, allQueuesHigh[t], label="High " + str(arrRates[t])
+ " us")

417     plt.legend(loc='best')
419     plt.xlabel('Time (seconds)')
421     plt.ylabel('Customers waiting in queue')
423     plt.title('Preemptive Varying arrRate High Priority')
425     plt.show()

427 if( doVaryingArrivalNoPriority == True):
429     print("
#####
print("##### Averaged No Priority varying arrRate
#####")
431     print("
#####")
433     allQueuesLow = []

```

```

413 allQueuesHigh = []
413 allQueuesComb = []

415 for l in range(0, len(arrRates)):
417     time = np.arange(0, 101*10**6, 1000000)

419     totalInterArrivalLow = []
419     totalInterArrivalHigh = []
421     totalTransmissionLow = []
421     totalTransmissionHigh = []
423     totalResponseLow = []
423     totalResponseHigh = []
425     totalAveragePacketSizeLow = []
425     totalAveragePacketSizeHigh = []
427     totalAverageDelaysLow = []
427     totalAverageDelaysHigh = []
429     totalArrivalRateLow = []
429     totalArrivalRateHigh = []
431     totalServiceRateLow = []
431     totalServiceRateHigh = []
433     allCombinedLengths = [[]] * averagedOver
433     allLengthsHighPriority = [[]] * averagedOver
435     allLengthsLowPriority = [[]] * averagedOver

437     for k in range(0, averagedOver):
439         packets = [[], []]
439         transCap = linkCapacity

441     # Read the csv file data into respective lists to store the
data
441     interArrivalHigh = list(np.random.exponential(arrRates[l] ,
441 int(100/(arrRates[l]*10**-6))))
443     packetSizeHigh = list(np.random.exponential(1000, int(100/(
443 arrRates[l]*10**-6))))
443     interArrivalLow = list(np.random.exponential(2000 , 50000))
445     packetSizeLow = list(np.random.exponential(1000, 50000))

447     # Create the list of low priority packets
447     for i in range(0, len(interArrivalLow)):
449         lowPacket = Packet(packetSizeLow[i], interArrivalLow[i],
LOW')
449         packets[0].append(lowPacket)

451     # Create the list of high priority packets
453     for i in range(0, len(interArrivalHigh)):
453         highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i
], 'HIGH')
455         packets[1].append(highPacket)

457     # Get the parameters
457     arrivalTimes , transTimes , queuingDelays , startEnd , respTimes ,
allPackets = calcNoPriority(transCap, packets)
459     sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))),
float(sum(packetSizeHigh))/len(packetSizeHigh)]

```

```

461         if (k == 0):
462             print("Link capacity -----",
transCap/10**6, "Mbps")
463             print("Number of low priority packets -----", len(
packetSizeLow), "packets")
464             print("Number of high priority packets -----", len(
packetSizeHigh), "packets")
465             print("Total number of packets -----", len(
packetSizeLow) + len(packetSizeHigh), "packets" )

466
467         print(l,k)
468         totalInterArrivalLow += interArrivalLow
469         totalInterArrivalHigh += interArrivalHigh
470         totalTransmissionLow += transTimes[0]
471         totalTransmissionHigh += transTimes[1]
472         totalResponseLow += respTimes[0]
473         totalResponseHigh += respTimes[1]
474         totalAveragePacketSizeLow += packetSizeLow
475         totalAveragePacketSizeHigh += packetSizeHigh
476         totalAverageDelaysLow += queuingDelays[0]
477         totalAverageDelaysHigh += queuingDelays[1]

478
479         startEnd = [[],[]]
480         arrivalTimes = []
481         startEndLow = [[],[]]
482         arrivalTimesLow = []
483         startEndHigh = [[],[]]
484         arrivalTimesHigh = []
485
486         for i in allPackets:
487             startEnd[0].append(i.startTime)
488             startEnd[1].append(i.endTime)
489             arrivalTimes.append(i.arrivalTime)

490
491             if (i.priority == 'LOW'):
492                 startEndLow[0].append(i.startTime)
493                 startEndLow[1].append(i.endTime)
494                 arrivalTimesLow.append(i.arrivalTime)
495
496             if (i.priority == 'HIGH'):
497                 startEndHigh[0].append(i.startTime)
498                 startEndHigh[1].append(i.endTime)
499                 arrivalTimesHigh.append(i.arrivalTime)

500
501         combined = []
502         lowPriority = []
503         highPriority = []

504
505
506         for i in range(0, len(time)):
507             length = 0
508             for j in range(0, len(arrivalTimesLow)):
509                 if (time[i] >= arrivalTimesLow[j] and startEndLow[1][
j] > time[i]):
510                     if (startEndLow[0][j] > arrivalTimesLow[j]):
511                         length += 1

```

```

lowPriority.append(length)

513
allLengthsLowPriority[k] = deepcopy(lowPriority)
515
    for i in range(0, len(time)):
517        length = 0
        for j in range(0, len(arrivalTimesHigh)):
519            if (time[i] >= arrivalTimesHigh[j] and startEndHigh
[1][j] > time[i]):
                    if (startEndHigh[0][j] > arrivalTimesHigh[j]):
521                        length += 1
                    highPriority.append(length)
523
allLengthsHighPriority[k] = deepcopy(highPriority)
525
allCombinedLengths[k] = list( map(add, highPriority,
lowPriority) )

527
    print("Average inter-arrival time -- (LOW) ----", round((sum(
totalInterArrivalLow))/(len(totalInterArrivalLow)), 4), "us" )
529
    print("Average inter-arrival time -- (HIGH) ----", round((sum(
totalInterArrivalHigh))/(len(totalInterArrivalHigh)), 4), "us" )
    print("Average inter-arrival time -- (TOTAL) --", round((sum(
totalInterArrivalHigh) + sum(totalInterArrivalLow))/(len(
totalInterArrivalHigh) + len(totalInterArrivalLow)), 4), "us" )
531
    print("Average transmission time --- (LOW) ----", round(sum(
totalTransmissionLow)/len(totalTransmissionLow), 4), "us" )
    print("Average transmission time --- (HIGH) ----", round(sum(
totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us" )
533
    print("Average transmission time --- (TOTAL) --", round((sum(
totalTransmissionHigh) + sum(totalTransmissionLow))/(len(
totalTransmissionHigh)+len(totalTransmissionLow)), 4), "us" )
    print("Average response time ----- (LOW) ----", round(sum(
totalResponseLow)/len(totalResponseLow), 4), "us" )
535
    print("Average response time ----- (HIGH) ----", round(sum(
totalResponseHigh)/len(totalResponseHigh), 4), "us" )
    print("Average response time ----- (TOTAL) --", round((sum(
totalResponseLow) + sum(totalResponseHigh))/(len(totalResponseLow) +
len(totalResponseHigh)), 4), "us" )
537
    print("The average packet size ----- (LOW) ----", round(sum(
totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow), 4), "bits"
)
    print("The average packet size ----- (HIGH) ----", round(sum(
totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh), 4), "
bits" )
539
    print("The average packet size ----- (TOTAL) --", round((sum(
totalAveragePacketSizeLow) + sum(totalAveragePacketSizeHigh)) / (float
(len(totalAveragePacketSizeLow)) + float(len(
totalAveragePacketSizeHigh))), 4), "bits" )
    print("Average delays ----- (LOW) ----", round(sum(
totalAverageDelaysLow)/len(totalAverageDelaysLow), 4), "us" )
541
    print("Average delays ----- (HIGH) ----", round(sum(
totalAverageDelaysHigh)/len(totalAverageDelaysHigh), 4), "us" )
    print("Average delays ----- (TOTAL) --", round((sum(
totalAverageDelaysLow) + sum(totalAverageDelaysHigh))/(len(
totalAverageDelaysHigh) + len(totalAverageDelaysLow)), 4), "us" )

```

```

543     print("Average arrival rate (lambda) (LOW) ----", round((len(
totalAveragePacketSizeLow))/((sum(totalInterArrivalLow))*10**6, 4), "
packets/second")
    print("Average arrival rate (lambda) (HIGH) ---", round((len(
totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh))*10**6, 4),
"packets/second")
545     print("Average arrival rate (lambda) (TOTAL) --", round((len(
totalAveragePacketSizeLow) + len(totalAveragePacketSizeHigh))/((sum(
totalInterArrivalHigh) + sum(totalInterArrivalLow))*10**6, 4), "
packets/second")
    print("Average service rate (mu) --- (LOW) ----", round(transCap
/(sum(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)), 4),
"packets/second")
547     print("Average service rate (mu) --- (HIGH) ---", round(transCap
/(sum(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)), 4)
, "packets/second")
    print("Average service rate (mu) --- (TOTAL) --", round(transCap
/((sum(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)*50000
+ sum(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)*
arrRates[1]*10**-6)/(50000+arrRates[1]*10**-6)), 4), "packets/second")
549
    time = time/10**6
551
    averagedCombinedLength = [0] * len(allCombinedLengths[0])
553    averagedLowPriorLength = [0] * len(allLengthsLowPriority[0])
    averagedHighPrioLength = [0] * len(allLengthsHighPriority[0])
555
    for x in range(0, len(allLengthsHighPriority)):
557        for y in range(0, len(allLengthsHighPriority[x])):
            averagedCombinedLength[y] += allCombinedLengths[x][y]
559            averagedLowPriorLength[y] += allLengthsLowPriority[x][y]
            averagedHighPrioLength[y] += allLengthsHighPriority[x][y]
561
    averagedCombinedLength = np.asarray(averagedCombinedLength)
563    averagedLowPriorLength = np.asarray(averagedLowPriorLength)
    averagedHighPrioLength = np.asarray(averagedHighPrioLength)
565
    for x in range(0, len(averagedCombinedLength)):
567        averagedCombinedLength[x] = averagedCombinedLength[x]/
averagedOver
569        averagedLowPriorLength[x] = averagedLowPriorLength[x]/
averagedOver
        averagedHighPrioLength[x] = averagedHighPrioLength[x]/
averagedOver
571
    allQueuesComb.append(deepcopy(averagedCombinedLength))
573    allQueuesHigh.append(deepcopy(averagedHighPrioLength))
    allQueuesLow.append(deepcopy(averagedLowPriorLength))
575
    for a in range(0, len(allQueuesLow)):
577        print("Average queue length low priority " + str(arrRates[a]),
round(sum(allQueuesLow[a])/len(allQueuesLow[a]), 4), "packets/second")
        print("Average queue length high priority " + str(arrRates[a])
, round(sum(allQueuesHigh[a])/len(allQueuesHigh[a]), 4), "packets/
second")

```

```

579 plt.figure()
581 for t in range(0, len(allQueuesHigh)):
    plt.plot(time, allQueuesLow[t], label="Low " + str(arrRates[t]) +
    " us")
583
585 plt.legend(loc='best')
587 plt.xlabel('Time (seconds)')
589 plt.ylabel('Customers waiting in queue')
591 plt.title('No priority Varying arrRate Low LowPriority')
593 plt.show()
595
597 plt.figure()
    plt.plot(time, allQueuesHigh[t], label="High " + str(arrRates[t])
    + " us")
593 plt.legend(loc='best')
595 plt.xlabel('Time (seconds)')
597 plt.ylabel('Customers waiting in queue')
    plt.title('No priority Varying arrRate High Priority')
    plt.show()

```

Code/DriverVary.py