

ERP 420

RESEARCH PROJECT

PRACTICAL 2: PRIORITY QUEUING

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

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.

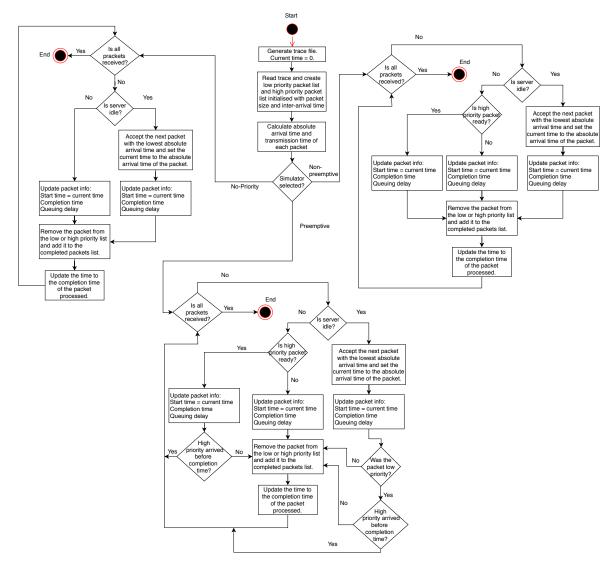


Figure 1. Flow diagram of the preemptive non-resume, non-preemptive and no priority simulators.

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:

Transmission time =
$$\frac{\text{Packet size}}{\text{Link capacity}}$$
. (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 interarrival 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

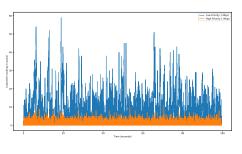
For this simulation, the same performance parameters will be calculated as in Section 3.5.1 and the queue length will also be plotted against time. The results will, however, be averaged out over 150 runs. The simulator will, therefore, generate the traces file 150 times, process the data and average 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.

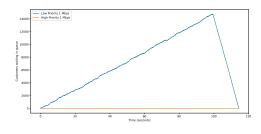


Treatment is

(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.

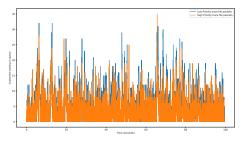


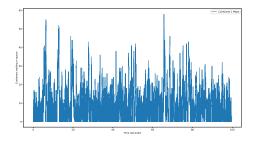
1400120

(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.

		Calculated for			
Simulator	Parameter	Low	High	Combined	Unit
Non-preemptive	Average	16340.423	2517.5917	10196.9424	μs
Preemptive	response	16727115.2507	1673.0544	9293585.3857	μs
No priority	time	10159.2036	10132.7128	10147.4299	μs
Non-preemptive	Average	15338.0252	1514.4253	9194.203	μs
Preemptive	queueing	16726112.8529	669.888	9292582.6463	μs
No priority	delays	9156.8058	9129.5464	9144.6905	μs
Non-preemptive	Queue	8.1513	0.9781	9.1294	packets
Preemptive	length	6969.2104	0.2299	6969.4403	packets
No priority	lengun	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
	Low	502.0532	packets/s
Arrival rate	High	402.1524	packets/s
	Combined	904.2053	packets/s
	Low	997.6079	packets/s
Service rate	High	996.8436	packets/s
	Combined	997.2681	packets/s

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

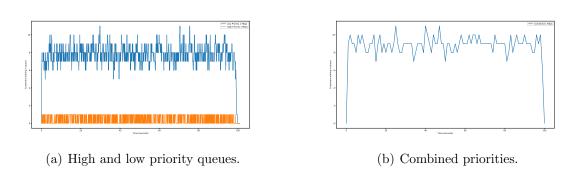
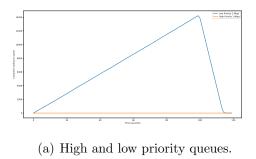
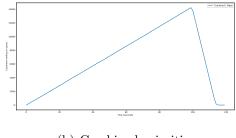


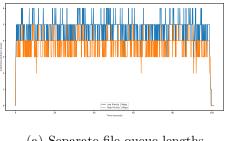
Figure 5. High and low priority queue lengths plotted against time separately 5(a) and combined 5(b) for a 1 Mbps non-preemptive priority simulator, averaged over 150 simulations.

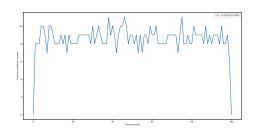




(b) Combined priorities.

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.

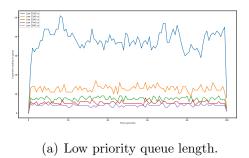
		Calculated for			
Simulator	Parameter	Low	High	Combined	Unit
Non-preemptive	Average	16807.1566	2511.3844	10453.48	μs
Preemptive	response	16520435.8291	1672.9732	9178763.4487	μs
No priority	time	10368.7603	10387.9942	10377.3087	μs
Non-preemptive	Average	15807.0084	1511.5727	9453.4814	μs
Preemptive	queueing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

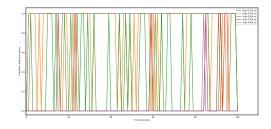
Table 4. Averaged performance parameters over 150 simulations, calculated for a 1 Mbps link capacity with 50000 low priority packets and 40000 high priority packets.

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.

			Packet pri	ority		
Parameter	Simulator	IAT	Low	High	Combined	Unit
	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
		2100	28022574	908	14353475	μs
Average		2300	21222463	764	11351937	μs
queuing	Preemptive	2500	16033262	666	8907664	μs
delay		2700	11617568	588	6674175	μs
		2900	7725751	526	4572639	μs
		2100	42152	42212	42181	μs
	No priority	2300	14155	14166	14160	μs
		2500	8889	8878	8884	μs
		2700	6694	6696	6695	μs
		2900	5366	5373	5369	μs
	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
Average		2300	8488	0.00	8488	packets
queue		2500	6413	0.00	6413	packets
length		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

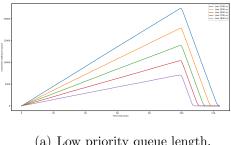
Table 5. Averaged performance parameters over 100 simulations, calculated for a 1 Mbps link capacity with varying the average inter-arrival times (IAT) in microseconds.

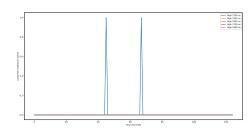




(b) High priority queue length.

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.

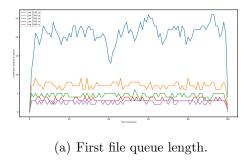


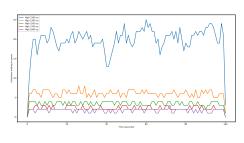


(a) Low priority queue length.

(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.





(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 gueues.

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, nonpreemptive 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 nonresume 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 -*-
  Created on Mon Aug 5 15:15:31 2019
  @author: project
  import csv
  import time
11
  class Packet:
       def __init__(self , size , interArrivalTime , priority):
13
           self.size
                                    = size
           self.interArrivalTime = interArrivalTime
           self.priority
                                    = priority
           self.arrivalTime
                                    = 0
           self.transTime
           self.queueDelay
19
           self.startTime
           self.endTime
                                    = 0
21
           self.responseTime
  # Read the text file and the inter-arrival time and the size
  def readCSV(filePath):
       arrival = []
       size = []
27
       with open(filePath, 'r') as f:
           data = csv.reader(f)
           for row in data:
31
                arrival.append(row[0])
                size.append(row[1])
33
      # Convert the strings lists to integer lists
35
       arrival = list(map(int, arrival))
       size = list(map(int, size))
37
       return arrival, size
39
41
  def getTraceInfo():
      # Read the csv file data into respective lists to store the data
43
       interArrivalHigh , packetSizeHigh = readCSV('HighPriority.txt')
       interArrivalLow , packetSizeLow = readCSV('LowPriority.txt')
45
      print("Low Priority Packets :", len(interArrivalLow))
print("High Priority Packets :", len(interArrivalHigh))
print("Avg interarrival Low :", sum(interArrivalLow)/len(
      interArrivalLow))
       print("Avg interarrival High :", sum(interArrivalHigh)/len(
49
      interArrivalHigh))
                                        :", sum(packetSizeLow)/len(packetSizeLow
       print ("Avg size Low
      ))
```

```
:", sum(packetSizeHigh)/len(
      print ("Avg size High
51
     packetSizeHigh))
      print("Total interarrival :", (sum(interArrivalHigh)+sum(
     interArrivalLow))/(len(interArrivalHigh)+len(interArrivalLow)))
 getTraceInfo()
55
57 # Takes the packets as a list of two lists corrensponding to
 # [[class:packetLowPriority], class[packetHighPriority]] and the
     transmission
59 # capacity as an integer and returns arrivalTimes, transTimes,
     queuingDelays,
  # startEnd, idleTimes, respTimes.
 def calcNonPreemptive(transCap, packets):
      currTime
                       = 0
      completedPackets = []
63
      # Calculate when the packets were received for low priority
65
      for i in range(0, len(packets[0])):
          if (i == 0):
67
               packets [0][i]. arrivalTime = packets [0][i]. interArrivalTime
69
              packets[0][i].arrivalTime = packets[0][i].interArrivalTime +
     packets [0][i - 1]. arrivalTime
      # Calculate when the packets were received for high priority
      for i in range (0, len (packets [1])):
          if (i == 0):
              packets [1][i]. arrivalTime = packets [1][i]. interArrivalTime
          else:
              packets [1][i]. arrivalTime = packets [1][i]. interArrivalTime +
79
     packets [1][i - 1].arrivalTime
          # Calculate how long each packet takes to transmit for low
81
     priority
      for i in range(0, len(packets[0])):
          packets [0][i].transTime = (packets [0][i].size/transCap)*10**6
      # Calculate how long each packet takes to transmit for high priority
85
      for i in range(0, len(packets[1])):
          packets [1][i].transTime = (packets [1][i].size/transCap)*10**6
89
      while (len(packets[0]) > 0 \text{ or } len(packets[1]) > 0):
          # For the first packet arriving
91
          toProcess = None
93
          # If a high priority packet was received
95
          if (len(packets[1]) > 0 and packets[1][0].arrivalTime <= currTime
     ):
              toProcess = packets[1][0]
97
              packets[1] = packets[1][1:]
```

```
99
          # If a low priority packet was received and no high priority
           elif (len(packets[0]) > 0 and packets[0][0]. arrivalTime \leq
      currTime):
               toProcess = packets[0][0]
               packets[0] = packets[0][1:]
          # If server is idle, get the next packet and update the packet to
          # the corresponding arrival time
           else:
107
               # If there are still packets to be processed of high and low
      priority
                 (len(packets[1]) > 0 \text{ and } len(packets[0]) > 0):
109
                    if (packets [1][0]. arrivalTime \le packets [0][0].
      arrivalTime):
                       toProcess = packets[1][0]
                       currTime = packets [1][0].arrivalTime
                       packets[1] = packets[1][1:]
                   else:
                       toProcess = packets[0][0]
                       currTime = packets [0][0]. arrivalTime
                       packets[0] = packets[0][1:]
               elif (len(packets[1]) > 0):
                   toProcess = packets[1][0]
121
                   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):
               # Process all the packets that has arrived
               toProcess.startTime = currTime
                                     = currTime + toProcess.transTime
               toProcess.endTime
133
               toProcess.queueDelay = currTime - toProcess.arrivalTime
               completedPackets.append(toProcess)
               currTime = toProcess.endTime
               toProcess = None
      # Calculate the reponse times for each of the packets
       for i in range (len (completed Packets)):
           completedPackets[i].responseTime = completedPackets[i].endTime -
141
      completedPackets[i].startTime + completedPackets[i].queueDelay
       arrivalTimes
143
       transTimes
       queuingDelays
                      =
       startEnd
       respTimes
149
       for i in range(0, len(completedPackets)):
```

```
if (completedPackets[i].priority = "HIGH"):
                arrivalTimes [1]. append (completedPackets [i]. arrivalTime)
                transTimes [1].append(completedPackets[i].transTime)
                queuingDelays [1]. append (completedPackets [i]. queueDelay)
                startEnd[1].append([completedPackets[i].startTime,
      completedPackets[i].endTime])
                respTimes [1].append(completedPackets[i].responseTime)
            elif (completedPackets[i].priority == "LOW"):
                arrivalTimes [0]. append (completedPackets [i]. arrivalTime)
159
                transTimes [0]. append (completedPackets [i]. transTime)
                queuingDelays [0]. append (completedPackets [i]. queueDelay)
161
                startEnd[0].append([completedPackets[i].startTime,
      completedPackets[i].endTime])
                respTimes [0]. append (completedPackets [i]. responseTime)
163
       return arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
165
      completedPackets
167 # Takes the packets as a list of two lists corrensponding to
    [[class:packetLowPriority], class[packetHighPriority]] and the
      transmission
169 # capacity as an integer and returns arrivalTimes, transTimes,
      queuingDelays,
  # startEnd, idleTimes, respTimes.
def calcPreemptive(transCap, packets):
       currTime
       completedPackets = []
       # Calculate when the packets were received for low priority
175
       for i in range (0, len (packets [0])):
           if (i == 0):
                packets [0][i]. arrivalTime = packets [0][i]. interArrivalTime
179
                packets [0][i]. arrivalTime = packets [0][i]. interArrivalTime +
      packets [0][i - 1]. arrivalTime
       # Calculate when the packets were received for high priority
183
       for i in range(0, len(packets[1])):
           if (i = 0):
185
                packets [1][i]. arrivalTime = packets [1][i]. interArrivalTime
           else:
                packets [1][i]. arrivalTime = packets [1][i]. interArrivalTime +
189
      packets [1][i - 1]. arrivalTime
       # Calculate how long each packet takes to transmit for low priority
       for i in range (0, len (packets [0])):
           packets [0][i].transTime = (packets [0][i].size/transCap)*10**6
193
       # Calculate how long each packet takes to transmit for high priority
195
       for i in range (0, len (packets [1])):
           packets [1][i].transTime = (packets [1][i].size/transCap)*10**6
197
       while (\operatorname{len}(\operatorname{packets}[0]) > 0 \text{ or } \operatorname{len}(\operatorname{packets}[1]) > 0):
```

```
# For the first packet arriving
           toProcess = None
           # If a high priority packet was received
203
           if (len(packets[1]) > 0 and packets[1][0].arrivalTime <= currTime
      ):
               toProcess = packets[1][0]
205
               packets[1] = packets[1][1:]
           # If a low priority packet was received and no high priority
           elif (len(packets[0]) > 0 and packets[0][0]. arrivalTime \leq
209
      currTime):
               toProcess = packets[0][0]
211
           # If server is idle, get the next packet and update the packet to
           # the corresponding arrival time
213
           else:
               # If there are still packets to be processed of high and low
215
      priority
               if (len(packets[1]) > 0 and len(packets[0]) > 0):
                    if (packets [1][0]. arrivalTime \le packets [0][0].
217
      arrivalTime):
                        toProcess = packets[1][0]
                       currTime = packets [1][0].arrivalTime
                        packets[1] = packets[1][1:]
221
                   else:
223
                        toProcess = packets[0][0]
                       currTime = packets [0][0].arrivalTime
               elif (len(packets[1]) > 0):
                   toProcess = packets[1][0]
                   currTime = packets [1][0].arrivalTime
229
                   packets[1] = packets[1][1:]
               elif (len(packets[0]) > 0):
                   toProcess = packets[0][0]
233
                   currTime = packets [0][0].arrivalTime
           if (toProcess != None):
               toProcess.startTime = currTime
                                    = currTime + toProcess.transTime
               toProcess.endTime
               toProcess.queueDelay = currTime - toProcess.arrivalTime
               currTime = toProcess.endTime
241
               # Process all the packets that has arrived
               if (toProcess.priority == "LOW"):
                    if (len(packets[1]) > 0 and currTime > packets[1][0].
      arrivalTime):
                       currTime = packets [1][0].arrivalTime
                        toProcess = None
                        continue
247
                   else:
249
                       packets[0] = packets[0][1:]
```

```
251
               completedPackets.append(toProcess)
253
      # Calculate the reponse times for each of the packets
       for i in range(len(completedPackets)):
255
           completedPackets[i].responseTime = completedPackets[i].endTime -
      completedPackets[i].startTime + completedPackets[i].queueDelay
257
       arrivalTimes
                         transTimes
259
       queuingDelays
                        [[],[]]
       startEnd
                      =
261
       respTimes
263
       for i in range (0, len (completed Packets)):
265
           if (completedPackets[i].priority == "HIGH"):
               arrivalTimes [1]. append (completedPackets [i]. arrivalTime)
267
               transTimes [1].append(completedPackets[i].transTime)
               queuingDelays [1]. append (completedPackets [i]. queueDelay)
269
               startEnd[1].append([completedPackets[i].startTime,
      completedPackets[i].endTime])
               respTimes [1]. append (completedPackets [i]. responseTime)
27
           elif (completedPackets[i].priority == "LOW"):
               arrivalTimes [0]. append (completedPackets [i]. arrivalTime)
               transTimes [0].append(completedPackets[i].transTime)
               queuingDelays [0]. append (completedPackets [i]. queueDelay)
               startEnd[0].append([completedPackets[i].startTime,
      completedPackets[i].endTime])
               respTimes [0]. append (completedPackets [i]. responseTime)
       return arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
      completedPackets
  # Takes the packets as a list of two lists corrensponding to
  # [[class:packetLowPriority], class[packetHighPriority]] and the
283
      transmission
  # capacity as an integer and returns arrivalTimes, transTimes,
      queuingDelays,
285 # startEnd, idleTimes, respTimes.
  def calcNoPriority(transCap, packets):
287
       currTime
       completedPackets = []
289
      # Calculate when the packets were received for low priority
291
       for i in range (0, len (packets [0])):
           if (i = 0):
293
               packets [0][i].arrivalTime = packets [0][i].interArrivalTime
               packets [0][i]. arrivalTime = packets [0][i]. interArrivalTime +
297
      packets [0][i - 1].arrivalTime
      # Calculate when the packets were received for high priority
```

```
for i in range(0, len(packets[1])):
            if (i == 0):
301
                packets [1][i]. arrivalTime = packets [1][i]. interArrivalTime
303
                packets [1][i]. arrivalTime = packets [1][i]. interArrivalTime +
305
      packets [1][i - 1]. arrivalTime
       # Calculate how long each packet takes to transmit for low priority
       for i in range (0, len(packets[0])):
           packets [0][i].transTime = (packets [0][i].size/transCap)*10**6
309
       # Calculate how long each packet takes to transmit for high priority
       for i in range (0, len (packets [1])):
           packets [1][i].transTime = (packets [1][i].size/transCap)*10**6
       while (len(packets[0]) > 0 \text{ or } len(packets[1]) > 0):
           # For the first packet arriving
           toProcess = None
317
            if (len(packets[1]) > 0 and len(packets[0]) > 0 and packets
319
      [1][0]. arrivalTime \leq currTime and packets [0][0]. arrivalTime \leq
      currTime):
                if (packets [1][0]. arrivalTime <= packets [0][0]. arrivalTime):
                    toProcess = packets[1][0]
                    packets[1] = packets[1][1:]
323
                else:
                    toProcess = packets[0][0]
                    packets[0] = packets[0][1:]
327
           # If a high priority packet was received
            elif (len(packets[1]) > 0 and packets[1][0].arrivalTime <=
329
      currTime):
                toProcess = packets[1][0]
                packets[1] = packets[1][1:]
331
           # If a low priority packet was received and no high priority
333
            elif (len(packets[0]) > 0 and packets[0][0]. arrivalTime \leq
      currTime):
                toProcess = packets[0][0]
335
                packets[0] = packets[0][1:]
337
           # If server is idle, get the next packet and update the packet to
           # the corresponding arrival time
339
               # If there are still packets to be processed of high and low
341
      priority
                if (\operatorname{len}(\operatorname{packets}[1]) > 0 and \operatorname{len}(\operatorname{packets}[0]) > 0):
                     if (packets [1] [0]. arrivalTime < packets [0] [0]. arrivalTime
343
      ):
                         toProcess = packets[1][0]
                         currTime = packets [1][0].arrivalTime
345
                         packets[1] = packets[1][1:]
347
```

```
else:
349
                        toProcess = packets[0][0]
                        currTime = packets [0][0].arrivalTime
351
                        packets [0] = packets [0][1:]
                elif (len(packets[1]) > 0):
                    toProcess = packets[1][0]
355
                    currTime = packets [1][0].arrivalTime
                    packets[1] = packets[1][1:]
                elif (len(packets[0]) > 0):
359
                    toProcess = packets[0][0]
                    currTime = packets [0][0].arrivalTime
                    packets[0] = packets[0][1:]
363
           if (toProcess != None):
               # Process all the packets that has arrived
               toProcess.startTime = currTime
                                      = currTime + toProcess.transTime
                toProcess.endTime
367
                toProcess.queueDelay = currTime - toProcess.arrivalTime
               completedPackets.append(toProcess)
               currTime = toProcess.endTime
                toProcess = None
371
      # Calculate the reponse times for each of the packets
       for i in range (len (completed Packets)):
           completedPackets[i].responseTime = completedPackets[i].endTime -
375
      completedPackets[i].startTime + completedPackets[i].queueDelay
       arrivalTimes
       transTimes
       queuingDelays
       startEnd
       respTimes
381
       for i in range (0, len (completed Packets)):
           if (completedPackets[i].priority = "HIGH"):
385
                arrivalTimes [1]. append (completedPackets [i]. arrivalTime)
               transTimes\ [1]\ .\ append\ (\ completedPackets\ [\ i\ ]\ .\ transTime\ )
                queuingDelays [1]. append (completedPackets [i]. queueDelay)
               startEnd[1].append([completedPackets[i].startTime,
389
      completedPackets[i].endTime])
               respTimes [1].append(completedPackets[i].responseTime)
391
           elif (completedPackets[i].priority == "LOW"):
               arrivalTimes [0]. append (completedPackets [i]. arrivalTime)
393
               transTimes [0]. append (completedPackets [i]. transTime)
                queuingDelays [0]. append (completedPackets [i]. queueDelay)
395
               startEnd[0].append([completedPackets[i].startTime,
      completedPackets[i].endTime])
               respTimes [0]. append (completedPackets [i]. responseTime)
397
       return arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
399
      completedPackets
```

B Trace File Driver

```
# -*- coding: utf-8 -*-
3 Created on Mon Sep 16 09:40:23 2019
 @author: project
  import matplotlib.pyplot as plt
  from PriorityQueuing import readCSV, Packet, calcNoPriority,
     calcNonPreemptive, calcPreemptive
  import numpy as np
11 import time as t
  from operator import add
13
  doNonPreemptiveTrace
                            = True
  doPreemptiveTrace
                            = False
                             = False
  doNoPriority
17
19 \left| linkCapacity = 1*10**6 \right|
  if (doNoPriority == True):
23
      print ("
     Processing no priority queuing
     ######** )
25
      packets = [[],[]]
27
      time = []
29
      transCap = linkCapacity
31
     # Read the csv file data into respective lists to store the data
      interArrivalHigh, packetSizeHigh = readCSV('HighPriority.txt')
33
      interArrivalLow , packetSizeLow = readCSV('LowPriority.txt')
     # Create the list of low priority packets
      for i in range(0, len(interArrivalLow)):
37
         lowPacket = Packet(packetSizeLow[i], interArrivalLow[i], 'LOW')
         packets [0]. append (lowPacket)
39
     # Create the list of high priority packets
41
      for i in range(0, len(interArrivalHigh)):
```

```
highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH
43
          ')
                  packets [1]. append (highPacket)
45
          # Get the parameters
           arrivalTimes, transTimes, queuingDelays, startEnd, respTimes, allPackets =
           calcNoPriority(transCap, packets)
           sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))), float(
         sum(packetSizeHigh))/len(packetSizeHigh)]
49
           print ("Link capacity -----", transCap/10**6,"
         Mbps")
           print ("Number of low priority packets ----", len (packet Size Low),
           "packets")
           print ("Number of high priority packets ----", len (packetSizeHigh)
          , "packets")
           print ("Total number of packets -----", len (packetSizeLow)
         + len(packetSizeHigh), "packets")
           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"\ )
           print("Average inter-arrival time -- (TOTAL) --", round((sum(
         interArrivalHigh) + sum(interArrivalLow))/(len(interArrivalHigh) + len
         (interArrivalLow)), 4), "us")
           print ("Average transmission time --- (LOW) ----", round (sum (
57
         transTimes[0])/len(transTimes[0]), 4), "us")
           print("Average transmission time --- (HIGH) ---", round(sum(
         transTimes[1])/len(transTimes[1]), 4), "us")
           print ("Average transmission time --- (TOTAL) --", round ((sum
59
         transTimes[1]) + sum(transTimes[0]))/(len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTimes[1])+len(transTi
         transTimes [0])), 4), "us")
           print ("Average response time ----- (LOW) ---", round (sum (respTimes
          [0])/len(respTimes[0]), 4), "us")
          print("Average response time ----- (HIGH) ---", round(sum(respTimes
61
          [1])/len(respTimes[1]), 4), "us")
           print("Average response time ----- (TOTAL) --", round((sum(
         respTimes[0]) + sum(respTimes[1]))/(len(respTimes[0]) + len(respTimes[0])
          [1])), 4) , "us")
          print ("The average packet size ---- (LOW) ----", round (size Avg [0],
63
         4) , "bits" )
          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")
           print ("Average delays ----- (LOW) ----", round (sum (
         queuingDelays [0]) /len (queuingDelays [0]), 4), "us")
           print ("Average delays ----- (HIGH) ---", round (sum (
67
         queuingDelays[1])/len(queuingDelays[1]), 4), "us")
          print ("Average delays ----- (TOTAL) --", round ((sum(
         queuingDelays [0]) + sum(queuingDelays [1]))/(len (queuingDelays [0]) +
         len (queuing Delays [1])), 4), "us")
           print("Average arrival rate (lambda) (LOW) ----", round((len(lambda) low)))
69
         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(
71
      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")
       print("Average service rate (mu) --- (HIGH) ---", round(transCap/
73
      sizeAvg[1], 4), "packets/second")
       print("Average service rate (mu) --- (TOTAL) --", round(transCap/(((
      sizeAvg[0]*50000 + sizeAvg[1]*40000)/90000)), 4), "packets/second")
       startEnd = [[],[]]
       arrivalTimes = []
77
       startEndLow = [[],[]]
       arrivalTimesLow = []
       startEndHigh = [[],[]]
       arrivalTimesHigh = []
81
       for i in allPackets:
83
           startEnd[0].append(i.startTime)
           startEnd[1].append(i.endTime)
85
           arrivalTimes.append(i.arrivalTime)
           if (i.priority == 'LOW'):
                startEndLow[0].append(i.startTime)
89
                startEndLow[1].append(i.endTime)
                arrivalTimesLow.append(i.arrivalTime)
91
           if (i.priority == 'HIGH'):
9.9
                startEndHigh[0].append(i.startTime)
                startEndHigh [1].append(i.endTime)
95
                arrivalTimesHigh.append(i.arrivalTime)
97
       time = np.arange(0, startEnd[1][-1], 10000)
       combined = []
99
       lowPriority = []
       highPriority = []
       for i in range(0, len(time)):
           length = 0
           for j in range(0, len(arrivalTimesLow)):
                if (time[i] >= arrivalTimesLow[j] and startEndLow[1][j] >
      time [i]):
                    if (startEndLow[0][j] > arrivalTimesLow[j]):
107
                        length += 1
           lowPriority.append(length)
109
       for i in range (0, len(time)):
111
           length = 0
           for j in range (0, len (arrivalTimesHigh)):
113
                if (time[i] >= arrivalTimesHigh[j] and startEndHigh[1][j] >
      time [i]):
                    if (startEndHigh[0][j] > arrivalTimesHigh[j]):
115
                        length += 1
```

```
high Priority . append (length)
117
      combined = list( map(add, highPriority, lowPriority) )
119
      print ("Average queue length (LOW) -----", round (sum (
121
     lowPriority)/len(lowPriority), 4), "packets/second")
      print ("Average queue length (HIGH) -----", round (sum (
     highPriority)/len(highPriority), 4), "packets/second")
      print ("Average queue length (TOTAL) -----", round (sum (combined
123
     )/len(combined), 4), "packets/second")
125
      time = time/10**6
127
      plt.figure()
      plt.plot(time, combined, label=str('Combined 1 Mbps'))
129
      plt.legend(loc='best')
      plt.xlabel('Time (seconds)')
      plt.ylabel('Customers waiting in queue')
      plt.title('No priority combined Trace')
      plt.show()
135
      plt.figure()
      plt.plot(time, lowPriority, label=str('Low Priority trace file
137
      plt.plot(time, highPriority, label=str('High Priority trace file
     packets'))
      plt.legend(loc='best')
139
      plt.xlabel('Time (seconds)')
plt.ylabel('Customers waiting in queue')
141
      plt.title('No priority low and high Trace')
      plt.show()
145
  if (doNonPreemptiveTrace = True):
      print ("
147
     Processing non-preemptive queuing
     ######" )
      print ("
     timeNow = t.time()
      packets = [[],[]]
      time = []
153
      transCap = linkCapacity
      # Read the csv file data into respective lists to store the data
157
      interArrivalHigh , packetSizeHigh = readCSV('HighPriority.txt')
      interArrivalLow , packetSizeLow = readCSV('LowPriority.txt')
      # Create the list of low priority packets
161
      for i in range (0, len(interArrivalLow)):
          lowPacket = Packet(packetSizeLow[i], interArrivalLow[i], 'LOW')
163
          packets [0]. append (lowPacket)
```

```
165
      # Create the list of high priority packets
      for i in range(0, len(interArrivalHigh)):
167
          highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH
      ')
          packets [1]. append (highPacket)
      # Get the parameters
171
      arrivalTimes, transTimes, queuingDelays, startEnd, respTimes, allPackets =
       calcNonPreemptive(transCap, packets)
      sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))), float(
173
      sum(packetSizeHigh))/len(packetSizeHigh)]
      print("Link capacity -----", transCap/10**6,"
175
      print("Number of low priority packets ----", len(packetSizeLow),
       "packets")
      print ("Number of high priority packets ----", len (packetSizeHigh)
      , "packets")
      print ("Total number of packets -----", len (packetSizeLow)
     + len(packetSizeHigh), "packets")
      print ("Average inter-arrival time -- (LOW) ----", round ((sum(
179
      interArrivalLow))/(len(interArrivalLow)), 4), "us")
      print("Average inter-arrival time -- (HIGH) ---", round((sum(
      interArrivalHigh))/(len(interArrivalHigh)),4), "us")
      print("Average inter-arrival time -- (TOTAL) --", round((sum(
181
      interArrivalHigh) + sum(interArrivalLow))/(len(interArrivalHigh) + len
      (interArrivalLow)), 4), "us")
      print ("Average transmission time --- (LOW) ----", round (sum (
      transTimes[0])/len(transTimes[0]), 4), "us")
      print("Average transmission time --- (HIGH) ---", round(sum(
183
      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")
      print("Average response time ----- (LOW) ---", round(sum(respTimes
185
      [0])/len(respTimes[0]), 4), "us")
      print ("Average response time ----- (HIGH) ---", round (sum (respTimes
      [1])/len(respTimes[1]), 4), "us")
      print ("Average response time ----- (TOTAL) --", round ((sum(
      respTimes[0]) + sum(respTimes[1]))/(len(respTimes[0]) + len(respTimes[0])
      [1])), 4) , "us")
      print ("The average packet size ---- (LOW) ----", round (sizeAvg[0],
      4) , "bits" )
      print ("The average packet size ---- (HIGH) ---", round (size Avg [1],
189
      4) , "bits" )
      print ("The average packet size ---- (TOTAL) --", round ((sum (
      packetSizeLow) + sum(packetSizeHigh)) / (float(len(packetSizeLow)) +
      float(len(packetSizeHigh))),4), "bits")
      print("Average delays ----- (LOW) ----", round(sum(
      queuingDelays [0]) /len (queuingDelays [0]), 4), "us")
      queuingDelays [1]) /len (queuingDelays [1]) , 4), "us") print ("Average delays ---
      print("Average delays ----- (TOTAL) --", round((sum(
193
      queuingDelays [0]) + sum(queuingDelays [1]))/(len(queuingDelays [0]) +
      len (queuing Delays [1])), 4), "us")
```

```
print("Average arrival rate (lambda) (LOW) ----", round((len()))
             packetSizeLow))/((sum(interArrivalLow)))*10**6, 4), "packets/second")
              print ("Average arrival rate (lambda) (HIGH) ---", round ((len (
195
             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")
              print ("Average service rate (mu) --- (LOW) ----", round (transCap/
197
             sizeAvg[0], 4), "packets/second")
              print("Average service rate (mu) --- (HIGH) ---", round(transCap/
             sizeAvg[1], 4), "packets/second")
               print ("Average service rate (mu) --- (TOTAL) --", round (transCap/(((Capacitan - Capacitan - Capac
             sizeAvg[0]*50000 + sizeAvg[1]*40000)/90000)), 4), "packets/second")
201
              startEnd = [[],[]]
              arrivalTimes = []
203
              startEndLow = [[],[]]
              arrivalTimesLow = []
205
              startEndHigh = [[],[]]
              arrivalTimesHigh = []
207
              for i in allPackets:
                       startEnd[0].append(i.startTime)
                       startEnd[1].append(i.endTime)
211
                       arrivalTimes.append(i.arrivalTime)
                       if (i.priority = 'LOW'):
                               startEndLow[0].append(i.startTime)
215
                               startEndLow[1].append(i.endTime)
                               arrivalTimesLow.append(i.arrivalTime)
                       if (i.priority == 'HIGH'):
219
                               startEndHigh[0].append(i.startTime)
                               startEndHigh[1].append(i.endTime)
                               arrivalTimesHigh.append(i.arrivalTime)
223
              time = np.arange (0, startEnd[1][-1], 10000)
              lowPriority = []
              highPriority = []
              for i in range (0, len(time)):
                       length = 0
                       for j in range(0, len(arrivalTimesLow)):
                               if (time[i] >= arrivalTimesLow[j] and startEndLow[1][j] >
231
             time [ i ] ):
                                         if (startEndLow[0][j] > arrivalTimesLow[j]):
                                                 length += 1
233
                       lowPriority.append(length)
              for i in range (0, len(time)):
                       length = 0
237
                       for j in range(0, len(arrivalTimesHigh)):
                                if (time[i] >= arrivalTimesHigh[j] and startEndHigh[1][j] >
             time [ i ] ):
```

```
if (startEndHigh [0][j] > arrivalTimesHigh [j]):
                      length += 1
          high Priority.append(length)
243
      combined = list( map(add, highPriority, lowPriority) )
      print ("Average queue length (LOW) -----", round (sum (
247
     lowPriority)/len(lowPriority), 4), "packets/second")
      print ("Average queue length (HIGH) -----", round (sum (
     highPriority)/len(highPriority), 4), "packets/second")
      print ("Average queue length (TOTAL) -----", round (sum (combined
249
     )/len(combined), 4), "packets/second")
251
      time = time/10**6
      plt.figure()
      plt.plot(time, combined, label=str('Combined 1 Mbps'))
      plt.legend(loc='best')
      plt.xlabel('Time (seconds)')
plt.ylabel('Customers waiting in queue')
      plt.title('Non-preemptive combined queues Trace')
259
      plt.show()
      plt.figure()
      plt.plot(time, lowPriority, label=str('Low Priority 1 Mbps'))
263
      plt.plot(time, highPriority, label=str('High Priority 1 Mbps'))
      plt.legend(loc='best')
      plt.xlabel('Time (seconds)')
      plt.ylabel ('Customers waiting in queue')
267
      plt.title('Non-preemptive low and high queues Trace')
      plt.show()
269
      print(t.time() - timeNow)
  if (doPreemptiveTrace == True):
273
      print ("
     Processing preemptive queuing
     <del>|||||||||</del>" )
      print ("
     timeNow = t.time()
      packets = [[],[]]
      time = []
279
      transCap = linkCapacity
      # Read the csv file data into respective lists to store the data
283
      interArrivalHigh , packetSizeHigh = readCSV('HighPriority.txt')
      interArrivalLow , packetSizeLow = readCSV('LowPriority.txt')
285
      # Create the list of low priority packets
287
      for i in range (0, len (interArrivalLow)):
          lowPacket = Packet(packetSizeLow[i], interArrivalLow[i], 'LOW')
```

```
packets [0]. append (lowPacket)
291
      # Create the list of high priority packets
       for i in range(0, len(interArrivalHigh)):
293
           highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH
      <sup>,</sup> )
           packets [1]. append (highPacket)
295
      # Get the parameters
297
       arrivalTimes, transTimes, queuingDelays, startEnd, respTimes, allPackets =
       calcPreemptive(transCap, packets)
       sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))), float(
299
      sum(packetSizeHigh))/len(packetSizeHigh)]
       print ("Link capacity -----", transCap/10**6,"
301
      Mbps")
       print ("Number of low priority packets ----", len (packetSizeLow),
       print("Number of high priority packets ----", len(packetSizeHigh)
303
      , "packets")
       print("Total number of packets -----", len(packetSizeLow)
      + len(packetSizeHigh), "packets")
       print ("Average inter-arrival time -- (LOW) ----", round ((sum(
305
      interArrivalLow))/(len(interArrivalLow)), 4), "us")
       print ("Average inter-arrival time -- (HIGH) ---", round ((sum(
      interArrivalHigh))/(len(interArrivalHigh))\ ,4)\ ,\ "us"\ )
       print("Average inter-arrival time -- (TOTAL) --", round((sum(
307
      interArrivalHigh) + sum(interArrivalLow))/(len(interArrivalHigh) + len
      (interArrivalLow)), 4), "us")
       print ("Average transmission time --- (LOW) ----", round (sum (
      transTimes[0])/len(transTimes[0]), 4), "us")
       print ("Average transmission time --- (HIGH) ---", round (sum (
309
      transTimes[1])/len(transTimes[1]), 4), "us")
       print ("Average transmission time --- (TOTAL) --", round ((sum(
      transTimes[1]) + sum(transTimes[0]))/(len(transTimes[1])+len(transTimes[1])
      transTimes[0])), 4), "us")
       print ("Average response time ----- (LOW) ---", round (sum (respTimes
311
      [0])/len(respTimes[0]), 4), "us")
       print ("Average response time ----- (HIGH) ---", round (sum (respTimes
      [1])/len(respTimes[1]), 4), "us")
       print("Average response time ----- (TOTAL) --", round((sum(
313
      respTimes[0]) + sum(respTimes[1]))/(len(respTimes[0]) + len(respTimes[0])
      [1])), 4) , "us")
       print ("The average packet size ---- (LOW) ----", round (size Avg [0],
      4) , "bits" )
      print ("The average packet size ---- (HIGH) ---", round (size Avg [1],
315
      4) , "bits" )
       print ("The average packet size ---- (TOTAL) --", round ((sum (
      packetSizeLow) + sum(packetSizeHigh)) / (float(len(packetSizeLow)) +
float(len(packetSizeHigh))),4), "bits")
       print ("Average delays ----- (LOW) ----", round (sum (
317
      queuingDelays[0])/len(queuingDelays[0]), 4), "us")
                                                         , round(sum(
       print ("Average delays ----- (HIGH) ---"
      queuingDelays [1]) / len (queuingDelays [1]) \;, \; \; 4) \;, \; "us")
       print("Average delays ----- (TOTAL) --", round((sum(
319
      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")
       print("Average arrival rate (lambda) (HIGH) ---", round((len(
321
      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")
       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")
       print("Average service rate (mu) --- (TOTAL) --", round(transCap/(((
325
      sizeAvg[0]*50000 + sizeAvg[1]*40000)/90000), 4), "packets/second"
       startEnd = [[],[]]
       arrivalTimes = []
329
       startEndLow = [[],[]]
       arrivalTimesLow = []
331
       startEndHigh = [[],[]]
       arrivalTimesHigh = []
333
       for i in allPackets:
           startEnd[0].append(i.startTime)
           startEnd[1].append(i.endTime)
337
           arrivalTimes.append(i.arrivalTime)
339
           if (i.priority == 'LOW'):
               startEndLow[0].append(i.startTime)
341
               startEndLow[1].append(i.endTime)
               arrivalTimesLow.append(i.arrivalTime)
343
           if (i.priority == 'HIGH'):
345
               startEndHigh[0].append(i.startTime)
               startEndHigh [1].append(i.endTime)
347
               arrivalTimesHigh.append(i.arrivalTime)
349
       time = np.arange(0, 120*10**6, 10000)
       lowPriority = []
351
       highPriority = []
353
       for i in range (0, len(time)):
355
           length = 0
           for j in range(0, len(arrivalTimesLow)):
357
               if (time[i] >= arrivalTimesLow[j] and startEndLow[0][j] >
      time[i] and startEndLow[0][j] > arrivalTimesLow[j]):
                     length += 1
350
           lowPriority.append(length)
361
       for i in range (0, len(time)):
           length = 0
363
           for j in range (0, len (arrivalTimesHigh)):
```

```
if (time[i] >= arrivalTimesHigh[j] and startEndHigh[0][j] >
365
      time[i] and startEndHigh[0][j] > arrivalTimesHigh[j]):
                       length += 1
           high Priority.append (length)
367
       combined = list( map(add, highPriority, lowPriority) )
       print ("Average queue length (LOW) ----", round (sum (
371
      lowPriority)/len(lowPriority), 4), "packets/second")
       print ("Average queue length (HIGH) -----", round (sum (
      highPriority)/len(highPriority), 4), "packets/second")
      print ("Average queue length (TOTAL) -----", round (sum (combined
373
      )/len(combined), 4), "packets/second")
       time = time/10**6
       plt.figure()
       plt.plot(time, combined, label=str('Combined 1 Mbps'))
379
       plt.legend(loc='best')
       plt.xlabel('Time (seconds)')
plt.ylabel('Customers waiting in queue')
       plt.title('Combined Preemptive queues Trace')
383
       plt.show()
       plt.figure()
       plt.plot(time, lowPriority, label=str('Low Priority 1 Mbps'))
387
       plt.plot(time, highPriority, label=str('High Priority 1 Mbps'))
       plt.legend(loc='best')
       plt.xlabel('Time (seconds)')
       plt.ylabel('Customers waiting in queue')
391
       plt.title('Preemptive queues low and high Trace')
393
       plt.show()
       print(t.time() - timeNow)
```

Code/DriverTrace.py

C Averaging Results Driver

```
14 do Average Non Preemptive
                             = False
  doAverageNoPriority
                             = True
16
  averagedOver = 150
 linkCapacity = 1*10**6
  if (doAverageNonPreemptive == True):
20
      print (
     Averaged Non-preemptive queuing
22
     <del>////////////////</del>")
     print("
     24
     time = np.arange(0, 101*10**6, 1000000)
26
      totalInterArrivalLow
28
      totalInterArrivalHigh
     total Transmission Low
30
     totalTransmissionHigh
     totalResponseLow
32
     totalResponseHigh
     total Average Packet Size Low\\
34
     totalAveragePacketSizeHigh =
     totalAverageDelaysLow
36
     totalAverageDelaysHigh
     totalArrivalRateLow
38
      totalArrivalRateHigh
      totalServiceRateLow
40
      totalServiceRateHigh
                                =
     allCombinedLengths
                                      * averagedOver
                                =
     allLengthsHighPriority
                                       * averagedOver
     allLengthsLowPriority
                                       * averagedOver
44
      for k in range (0, averagedOver):
46
          print(k)
          packets = [[],[]]
48
         transCap = linkCapacity
         # Read the csv file data into respective lists to store the data
         interArrivalHigh = list (np.random.exponential (2486, 40000))
         packetSizeHigh = list(np.random.exponential(1000, 40000))
         interArrivalLow = list(np.random.exponential(1991, 50000))
          packetSizeLow
                          = list (np.random.exponential (1000, 50000))
56
         # Create the list of low priority packets
         for i in range(0, len(interArrivalLow)):
             lowPacket = Packet(packetSizeLow[i], interArrivalLow[i], 'LOW
     ')
             packets [0]. append (lowPacket)
         # Create the list of high priority packets
62
         for i in range(0, len(interArrivalHigh)):
             \label{eq:high-packet} high\,[\,i\,]\,,\ interArrivalHigh\,[\,i\,]\,,
     HIGH')
```

```
packets [1]. append (highPacket)
66
          # Get the parameters
           arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
68
      allPackets = calcNonPreemptive(transCap, packets)
           sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))),
      float (sum(packetSizeHigh))/len(packetSizeHigh)]
70
           if (k == 0):
               print ("Link capacity -----", transCap
72
      /10**6, "Mbps")
               print ("Number of low priority packets ----", len (
      packetSizeLow), "packets")
               print ("Number of high priority packets -----", len (
74
      packetSizeHigh), "packets")
               print ("Total number of packets -----", len (
      packetSizeLow) + len(packetSizeHigh), "packets" )
76
           totalInterArrivalLow
                                      += interArrivalLow
                                      += interArrivalHigh
           totalInterArrivalHigh
           totalTransmissionLow
                                      += transTimes [0]
           totalTransmissionHigh
                                      += transTimes [1]
80
           totalResponseLow
                                      += respTimes[0]
           totalResponseHigh
                                      += respTimes[1]
           total Average Packet Size Low
                                      += packetSizeLow
           totalAveragePacketSizeHigh += packetSizeHigh
           totalAverageDelaysLow
                                      += queuingDelays [0]
           totalAverageDelaysHigh
                                      += queuingDelays[1]
           startEnd = [[],[]]
88
           arrivalTimes = []
           startEndLow = [[],[]]
           arrivalTimesLow = []
           startEndHigh = [[],[]]
92
           arrivalTimesHigh = []
           for i in allPackets:
               startEnd[0].append(i.startTime)
96
               startEnd[1].append(i.endTime)
               arrivalTimes.append(i.arrivalTime)
               if (i.priority == 'LOW'):
100
                   startEndLow[0].append(i.startTime)
                   startEndLow[1].append(i.endTime)
                   arrivalTimesLow.append(i.arrivalTime)
               if (i.priority == 'HIGH'):
                   startEndHigh[0].append(i.startTime)
                   startEndHigh [1].append(i.endTime)
                   arrivalTimesHigh.append(i.arrivalTime)
108
           combined = []
110
           lowPriority = []
           highPriority = []
112
114
```

```
for i in range (0, len(time)):
               length = 0
116
               for j in range (0, len (arrivalTimesLow)):
                   if (time[i] >= arrivalTimesLow[j] and startEndLow[1][j] >
118
       time [ i ] ):
                       if (startEndLow[0][j] > arrivalTimesLow[j]):
                           length += 1
120
               lowPriority.append(length)
          allLengthsLowPriority[k] = deepcopy(lowPriority)
124
          for i in range (0, len(time)):
               length = 0
               for j in range (0, len (arrivalTimesHigh)):
                   if (time[i] >= arrivalTimesHigh[j] and startEndHigh[1][j]
128
      > time[i]):
                       if (startEndHigh[0][j] > arrivalTimesHigh[j]):
                           length += 1
130
               high Priority.append (length)
132
           allLengthsHighPriority[k] = deepcopy(highPriority)
          allCombinedLengths[k] = list(map(add, highPriority, lowPriority)
134
       )
      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"
      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")
      print("Average transmission time --- (HIGH) ---", round(sum(
140
      totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us")
      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")
      print("Average response time ----- (HIGH) ---", round(sum(
      totalResponseHigh)/len(totalResponseHigh), 4), "us")
      print("Average response time ----- (TOTAL) --", round((sum(
144
      totalResponseLow) + sum(totalResponseHigh))/(len(totalResponseLow) +
      len(totalResponseHigh)), 4) , "us" )
      print("The average packet size ----", round(sum(
      totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow), 4), "bits"
      print ("The average packet size ---- (HIGH) ---", round (sum (
146
      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 (
148
      totalAverageDelaysLow)/len(totalAverageDelaysLow), 4), "us")
      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")
      print("Average arrival rate (lambda) (LOW) ----", round((len(
      totalAveragePacketSizeLow))/((sum(totalInterArrivalLow)))*10**6, 4), "
      packets/second")
      print ("Average arrival rate (lambda) (HIGH) ---", round ((len (
152
      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")
      print ("Average service rate (mu) --- (LOW) ----", round (transCap/(sum
154
      (totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)), 4),
      packets/second")
      print ("Average service rate (mu) --- (HIGH) ---", round (transCap/(sum
      (totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)), 4),
      packets/second")
      print("Average service rate (mu) --- (TOTAL) --", round(transCap/((
156
      sum(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)*50000 +
      sum(totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)*50000)
      /90000), 4), "packets/second")
      time = time/10**6
160
      averagedCombinedLength = [0] * len(allCombinedLengths[0])
      averagedLowPriorLength = [0] * len(allLengthsLowPriority[0])
162
      averagedHighPrioLength = [0] * len(allLengthsHighPriority[0])
164
      for x in range(0, len(allLengthsHighPriority)):
           for y in range (0, len (allLengthsHighPriority[x])):
166
               averagedCombinedLength[y] += allCombinedLengths[x][y]
               averagedLowPriorLength[y] += allLengthsLowPriority[x][y]
168
               averagedHighPrioLength[y] += allLengthsHighPriority[x][y]
170
      averagedCombinedLength = np.asarray(averagedCombinedLength)
      averagedLowPriorLength = np.asarray(averagedLowPriorLength)
      averaged High PrioLength \, = \, np.\, as array \, (\, averaged High PrioLength \, )
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/
```

```
second")
      print ("Average queue length (HIGH) ----", round (sum (
182
      averagedHighPrioLength)/len(averagedHighPrioLength), 4), "packets/
      second")
      print("Average queue length (TOTAL) -----", round(sum(
      averagedCombinedLength)/len(averagedCombinedLength), 4), "packets/
      second")
184
      plt.figure()
186
      plt.plot(time, averagedCombinedLength, label=str('Combined 1 Mbps'))
      plt.legend(loc='best')
188
      plt.xlabel('Time (seconds)')
plt.ylabel('Customers waiting in queue')
190
      plt.title('Averaged combined non-preemptive queues')
      plt.show()
192
      plt.figure()
194
      plt.plot(time, averagedLowPriorLength, label=str('Low Priority 1 Mbps
      '))
      plt.plot(time, averagedHighPrioLength, label=str('High Priority 1 \\
     Mbps '))
      plt.legend(loc='best')
      plt.xlabel('Time (seconds)')
198
      plt.ylabel('Customers waiting in queue')
      plt.title ('Averaged low and high non-preemptive queues')
200
      plt.show()
202
  if (doAveragePreemptive = True):
204
      print (
     <del>###################################</del>*)
      Averaged Preemptive queuing
206
     ######" )
      print("
     208
      time = np.arange(0, 120*10**6, 1000000)
      totalInterArrivalLow
212
      totalInterArrivalHigh
      total Transmission Low
      totalTransmissionHigh
      totalResponseLow
216
      totalResponseHigh
      total Average Packet Size Low
218
      totalAveragePacketSizeHigh =
      totalAverageDelaysLow
220
      totalAverageDelaysHigh
      totalArrivalRateLow
      totalArrivalRateHigh
      totalServiceRateLow
224
      totalServiceRateHigh
                                 =
      allCombinedLengths
                                 = [[]]
                                        * averagedOver
226
      allLengthsHighPriority
                                        * averagedOver
                                 = [[]]
```

```
allLengthsLowPriority
                                  = [[]] * averagedOver
228
       for k in range(0, averagedOver):
230
           print(k)
           packets = [[],[]]
           transCap = linkCapacity
234
          # Read the csv file data into respective lists to store the data
           interArrivalHigh = list(np.random.exponential(2486, 40000))
                            = list (np.random.exponential (1000, 40000))
           packetSizeHigh
           interArrivalLow = list(np.random.exponential(1991, 50000))
238
                            = list (np.random.exponential (1000, 50000))
           packetSizeLow
          # Create the list of low priority packets
           for i in range(0, len(interArrivalLow)):
242
               lowPacket = Packet(packetSizeLow[i], interArrivalLow[i], 'LOW
      ')
               packets [0]. append (lowPacket)
244
          # Create the list of high priority packets
246
           for i in range(0, len(interArrivalHigh)):
               highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i],
248
     HIGH')
               packets [1]. append (highPacket)
          # Get the parameters
           arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
252
      allPackets = calcPreemptive(transCap, packets)
           sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))),
      float (sum(packetSizeHigh))/len(packetSizeHigh)]
           if (k == 0):
               print ("Link capacity -----", transCap
256
      /10**6, "Mbps")
               print ("Number of low priority packets ----", len (
      packetSizeLow), "packets")
               print ("Number of high priority packets -----", len (
258
      packetSizeHigh), "packets")
               print ("Total number of packets -----", len (
      packetSizeLow) + len(packetSizeHigh), "packets" )
260
           totalInterArrivalLow
                                      += interArrivalLow
           totalInterArrivalHigh
                                      += interArrivalHigh
           totalTransmissionLow
                                      += transTimes [0]
           totalTransmissionHigh
                                      += transTimes [1]
264
           totalResponseLow
                                      += respTimes[0]
           totalResponseHigh
                                      += respTimes[1]
266
           totalAveragePacketSizeLow
                                      += packetSizeLow
           totalAveragePacketSizeHigh += packetSizeHigh
268
           totalAverageDelaysLow
                                      += queuingDelays [0]
           totalAverageDelaysHigh
                                      += queuingDelays [1]
           startEnd = [[],[]]
272
           arrivalTimes = []
           startEndLow = [[],[]]
           arrivalTimesLow = []
```

```
startEndHigh = [[],[]]
276
           arrivalTimesHigh = []
278
           for i in allPackets:
               startEnd[0].append(i.startTime)
               startEnd[1].append(i.endTime)
               arrivalTimes.append(i.arrivalTime)
282
               if (i.priority = 'LOW'):
                   startEndLow[0].append(i.startTime)
                   startEndLow[1].append(i.endTime)
286
                   arrivalTimesLow.append(i.arrivalTime)
               if (i.priority = 'HIGH'):
                   startEndHigh[0].append(i.startTime)
290
                   startEndHigh[1].append(i.endTime)
                   arrivalTimesHigh.append(i.arrivalTime)
           combined = []
294
           lowPriority = []
           highPriority = []
298
           for i in range (0, len(time)):
               length = 0
               for j in range(0, len(arrivalTimesLow)):
                   if (time[i] >= arrivalTimesLow[j] and startEndLow[1][j] >
302
       time [ i ] ):
                        if (startEndLow[0][j] > arrivalTimesLow[j]):
                            length += 1
304
               lowPriority.append(length)
           allLengthsLowPriority[k] = deepcopy(lowPriority)
308
           for i in range(0, len(time)):
               length = 0
               for j in range (0, len (arrivalTimesHigh)):
                   if (time[i] >= arrivalTimesHigh[j] and startEndHigh[1][j]
312
       > time[i]):
                        if (\text{startEndHigh}[0][j] > \text{arrivalTimesHigh}[j]):
                            length += 1
               high Priority.append (length)
           allLengthsHighPriority[k] = deepcopy(highPriority)
           allCombinedLengths[k] = list(map(add, highPriority, lowPriority)
318
       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")
       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")
```

```
print ("Average transmission time --- (HIGH) ---", round (sum (
324
      totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us")
       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")
       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" )
       print ("The average packet size ---- (LOW) ----", round (sum (
      totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow), 4),
       print ("The average packet size ---- (HIGH) ---", round (sum (
330
      totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh), 4),
       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 (
332
      totalAverageDelaysLow)/len(totalAverageDelaysLow), 4), "us")
       print ("Average delays ----- (HIGH) ---", round (sum (
      totalAverageDelaysHigh)/len(totalAverageDelaysHigh), 4), "us")
       print ("Average delays ----- (TOTAL) --", round ((sum(
334
      totalAverageDelaysLow) + sum(totalAverageDelaysHigh))/(len(
      total Average Delays High) \ + \ len (total Average Delays Low)), \ 4), \ "us")
       print ("Average arrival rate (lambda) (LOW) ----", round ((len (
      totalAveragePacketSizeLow))/((sum(totalInterArrivalLow)))*10**6, 4), "
      packets/second")
       print ("Average arrival rate (lambda) (HIGH) ---", round ((len (
336
      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")
       print("Average service rate (mu) --- (LOW) ----", round(transCap/(sum
338
      (totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)), 4),
      packets/second")
       print ("Average service rate (mu) --- (HIGH) ---", round (transCap/(sum
      (totalAveragePacketSizeHigh)/len(totalAveragePacketSizeHigh)), 4),
      packets/second")
       print("Average service rate (mu) --- (TOTAL) --", round(transCap/((
340
      sum(totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)*50000 +
      sum (total Average Packet Size High) / len (total Average Packet Size High) *50000)
      /90000), 4), "packets/second")
       time = time/10**6
342
       averagedCombinedLength = [0] * len(allCombinedLengths[0])
       averagedLowPriorLength = [0] * len(allLengthsLowPriority[0])
344
       averaged High PrioLength \ = \ [0] \ * \ len(all Lengths High Priority \ [0])
       for x in range (0, len (allLengthsHighPriority)):
```

```
for y in range(0, len(allLengthsHighPriority[x])):
348
              averagedCombinedLength[y] += allCombinedLengths[x][y]
              averagedLowPriorLength[y] += allLengthsLowPriority[x][y]
350
              averagedHighPrioLength[y] += allLengthsHighPriority[x][y]
      averagedCombinedLength = np.asarray(averagedCombinedLength)
      averagedLowPriorLength = np.asarray(averagedLowPriorLength)
354
      averagedHighPrioLength = np.asarray(averagedHighPrioLength)
356
      for x in range (0, len (averagedCombinedLength)):
358
          averagedCombinedLength[x] = averagedCombinedLength[x]/
     averagedOver
          averagedLowPriorLength[x] = averagedLowPriorLength[x]/
360
     averagedOver
          averagedHighPrioLength[x] = averagedHighPrioLength[x]/
     averagedOver
362
      print ("Average queue length (LOW) -----", round (sum (
     averagedLowPriorLength)/len(averagedLowPriorLength), 4), "packets/
     second")
      print ("Average queue length (HIGH) -----", round (sum (
364
     averagedHighPrioLength)/len(averagedHighPrioLength), 4), "packets/
      print ("Average queue length (TOTAL) -----", round (sum (
     averagedCombinedLength)/len(averagedCombinedLength), 4), "packets/
     second")
366
      plt.figure()
368
      plt.plot(time, averagedCombinedLength, label=str('Combined 1 Mbps'))
      plt.legend(loc='best')
370
      plt.xlabel('Time (seconds)')
      plt.ylabel('Customers waiting in queue')
372
      plt.title('Averaged combined preemptive queues')
      plt.show()
      plt.figure()
      plt.plot(time, averagedLowPriorLength, label=str('Low Priority 1 Mbps
      '))
      plt.plot(time, averagedHighPrioLength, label=str('High Priority 1
378
     Mbps'))
      plt.legend(loc='best')
      plt.xlabel('Time (seconds)')
plt.ylabel('Customers waiting in queue')
      plt.title('Averaged low and high preemptive queues')
382
      plt.show()
  if (doAverageNoPriority = True):
386
      print ("
     Averaged No Priority queuing
388
     <del>#####</del>" )
      print ("
```

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

```
print ("Number of low priority packets ----", len (
440
      packetSizeLow), "packets")
               print ("Number of high priority packets ----", len (
      packetSizeHigh), "packets")
               print ("Total number of packets -----
      packetSizeLow) + len(packetSizeHigh), "packets" )
           totalInterArrivalLow
                                       += interArrivalLow
444
           totalInterArrivalHigh
                                        += interArrivalHigh
           totalTransmissionLow
                                       += transTimes [0]
446
           totalTransmissionHigh
                                       += transTimes[1]
           totalResponseLow
                                       += respTimes[0]
448
                                       += respTimes[1]
           totalResponseHigh
           total Average Packet Size Low\\
                                       += packetSizeLow
450
           totalAveragePacketSizeHigh += packetSizeHigh
           total Average Delays Low
                                       += queuingDelays [0]
452
                                       += queuingDelays[1]
           totalAverageDelaysHigh
454
           startEnd = [[],[]]
           arrivalTimes = []
456
           startEndLow = [[],[]]
           arrivalTimesLow = []
458
           startEndHigh = [[],[]]
           arrivalTimesHigh = []
           for i in allPackets:
462
               startEnd[0].append(i.startTime)
               startEnd[1].append(i.endTime)
               arrivalTimes.append(i.arrivalTime)
466
               if (i.priority = 'LOW'):
                   startEndLow[0].append(i.startTime)
                   startEndLow[1].append(i.endTime)
                   arrivalTimesLow.append(i.arrivalTime)
470
               if (i.priority = 'HIGH'):
                   startEndHigh [0].append(i.startTime)
                   startEndHigh[1].append(i.endTime)
474
                   arrivalTimesHigh.append(i.arrivalTime)
           combined = []
           lowPriority = []
478
           highPriority = []
           for i in range (0, len(time)):
               length = 0
482
               for j in range (0, len (arrivalTimesLow)):
                   if (time[i] >= arrivalTimesLow[j] and startEndLow[1][j] >
       time [ i ] ):
                        if (startEndLow[0][j] > arrivalTimesLow[j]):
                            length += 1
               lowPriority.append(length)
488
           allLengthsLowPriority[k] = deepcopy(lowPriority)
           for i in range (0, len(time)):
```

```
length = 0
492
               for j in range(0, len(arrivalTimesHigh)):
                   if (time[i] >= arrivalTimesHigh[j] and startEndHigh[1][j]
494
       > time[i]):
                        if (\operatorname{startEndHigh}[0][j] > \operatorname{arrivalTimesHigh}[j]):
                            length += 1
496
               high Priority.append (length)
498
           allLengthsHighPriority [k] = deepcopy (highPriority)
           allCombinedLengths[k] = list(map(add, highPriority, lowPriority)
500
       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"
       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")
       print ("Average transmission time --- (HIGH) ---", round (sum (
506
      totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us")
       print("Average transmission time --- (TOTAL) --", round((sum(
      totalTransmissionHigh) + sum(totalTransmissionLow))/(len(
      totalTransmissionHigh)+len(totalTransmissionLow)), 4), "us"
       print ("Average response time ----- (LOW) ---", round (sum (
508
      totalResponseLow)/len\left(totalResponseLow\right),~4)~,~"us"~)
       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" )
       print("The average packet size ----", round(sum())
      totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow), 4) ,
       )
       print ("The average packet size ---- (HIGH) ---", round (sum (
512
      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")
       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")
       print ("Average arrival rate (lambda) (LOW) ----", round ((len (
      totalAveragePacketSizeLow))/((sum(totalInterArrivalLow)))*10**6, 4), "
      packets/second")
       print ("Average arrival rate (lambda) (HIGH) ---", round ((len (
518
      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")
      print ("Average service rate (mu) --- (LOW) ----", round (transCap/(sum
520
      (totalAveragePacketSizeLow)/len(totalAveragePacketSizeLow)), 4),
      packets/second")
      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)*50000)
      /90000), 4), "packets/second")
      time = time/10**6
      averagedCombinedLength = [0] * len(allCombinedLengths[0])
      averagedLowPriorLength = [0] * len(allLengthsLowPriority[0])
526
      averagedHighPrioLength = [0] * len(allLengthsHighPriority[0])
       for x in range (0, len (allLengthsHighPriority)):
           for y in range (0, len (allLengthsHighPriority[x])):
530
               averagedCombinedLength\left[\,y\,\right] \,\,+\!\!=\,\, allCombinedLengths\left[\,x\,\right]\left[\,y\,\right]
               averagedLowPriorLength[y] += allLengthsLowPriority[x][y]
532
               averagedHighPrioLength[y] += allLengthsHighPriority[x][y]
      averagedCombinedLength = np.asarray(averagedCombinedLength)
      averagedLowPriorLength = np.asarray(averagedLowPriorLength)
536
      averagedHighPrioLength = np.asarray(averagedHighPrioLength)
      for x in range (0, len (averagedCombinedLength)):
540
           averagedCombinedLength[x] = averagedCombinedLength[x]/
      averagedOver
           averagedLowPriorLength[x] = averagedLowPriorLength[x]/
542
      averagedOver
           averagedHighPrioLength[x] = averagedHighPrioLength[x]/
      averagedOver
544
      print ("Average queue length (LOW) -----", round (sum (
      averagedLowPriorLength)/len(averagedLowPriorLength), 4), "packets/
      second")
      print ("Average queue length (HIGH) -----", round (sum (
546
      averagedHighPrioLength)/len(averagedHighPrioLength), 4), "packets/
      second")
      print("Average queue length (TOTAL) -----", round(sum(
      averagedCombinedLength)/len(averagedCombinedLength), 4), "packets/
      second")
548
      plt.figure()
550
       plt.plot(time, averagedCombinedLength, label=str('Combined 1 Mbps'))
       plt.legend(loc='best')
       plt.xlabel('Time (seconds)')
       plt.ylabel('Customers waiting in queue')
       plt.title('Averaged combined no-priority queues')
```

```
plt.show()

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

Code/DriverAverage.py

D Averaging Varying Arrival Rates Driver

```
\# -*- coding: utf-8 -*-
3 Created on Mon Sep 16 09:41:12 2019
 @author: project
 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 do Varying Arrival No Priority
                         = False
|averagedOver| = 5
 print("Averaged over:", averagedOver)
 arrRates = [2000] # Actually interArrivalTimes
 linkCapacity = 1*10**6
21
 if (doVaryingArrivalNonPreempt = True):
23
     print (
    Averaged Non-Preemptive varying arrRate
    <del>#####</del>")
     print ("
    allQueuesLow = []
     allQueuesHigh = []
29
     allQueuesComb = []
31
     for 1 in range (0, len(arrRates)):
```

```
time = np.arange(0, 101*10**6, 1000000)
33
35
          total Inter Arrival Low\\
          totalInterArrivalHigh
          totalTransmissionLow
          totalTransmissionHigh
39
          totalResponseLow
          totalResponseHigh
41
          totalAveragePacketSizeLow
          totalAveragePacketSizeHigh =
43
          totalAverageDelaysLow
          totalAverageDelaysHigh
          totalArrivalRateLow
          totalArrivalRateHigh
47
          total Service Rate Low\\
          totalServiceRateHigh
                                      =
          allCombinedLengths
                                             * averagedOver
                                        allLengthsHighPriority
                                      =
                                             * averagedOver
          allLengthsLowPriority
                                             * averagedOver
          for k in range(0, averagedOver):
              packets = [[],[]]
              transCap = linkCapacity
              print(1,k)
              # Read the csv file data into respective lists to store the
     data
              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))))
              interArrivalLow = list (np.random.exponential (2000, 50000))
61
              packetSizeLow
                               = list (np.random.exponential (1000, 50000))
63
              # Create the list of low priority packets
              for i in range(0, len(interArrivalLow)):
65
                  lowPacket = Packet(packetSizeLow[i], interArrivalLow[i],
      'LOW')
                  packets [0]. append (lowPacket)
67
              # Create the list of high priority packets
69
              for i in range (0, len (interArrivalHigh)):
                  highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i
     ], 'HIGH')
                  packets [1]. append (highPacket)
73
              # Get the parameters
              arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
75
     allPackets = calcNonPreemptive(transCap, packets)
              sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))),
     float (sum(packetSizeHigh))/len(packetSizeHigh)]
              if (k == 0):
77
                  print ("Link capacity ----",
     transCap/10**6, "Mbps")
                  print ("Number of low priority packets ----", len (
     packetSizeLow), "packets")
```

```
print ("Number of high priority packets -----", len (
      packetSizeHigh), "packets")
                   print ("Total number of packets -----
81
      packetSizeLow) + len(packetSizeHigh), "packets")
               totalInterArrivalLow
                                            += interArrivalLow
83
               totalInterArrivalHigh
                                            += interArrivalHigh
               total Transmission Low\\
                                            += transTimes [0]
85
               totalTransmissionHigh
                                            += transTimes [1]
               totalResponseLow
                                            += respTimes[0]
87
               totalResponseHigh
                                            += respTimes [1]
               total Average Packet Size Low
                                           += packetSizeLow
               totalAveragePacketSizeHigh += packetSizeHigh
               total Average Delays Low
                                            += queuingDelays [0]
91
               totalAverageDelaysHigh
                                            += queuingDelays [1]
93
               startEnd = [[],[]]
               arrivalTimes = []
95
               startEndLow = [[],[]]
               arrivalTimesLow = []
97
               startEndHigh = [[],[]]
               arrivalTimesHigh = []
99
               for i in allPackets:
                   startEnd[0].append(i.startTime)
                   startEnd[1].append(i.endTime)
103
                   arrivalTimes.append(i.arrivalTime)
105
                    if (i.priority = 'LOW'):
                        startEndLow[0].append(i.startTime)
                        startEndLow[1].append(i.endTime)
                        arrivalTimesLow.append(i.arrivalTime)
                   if (i.priority = 'HIGH'):
                        startEndHigh[0].append(i.startTime)
                        startEndHigh[1].append(i.endTime)
                        arrivalTimesHigh.append(i.arrivalTime)
               lowPriority = []
               high Priority = []
               for i in range (0, len(time)):
                   length = 0
                    for j in range (0, len (arrivalTimesLow)):
                        if (time[i] >= arrivalTimesLow[j] and startEndLow[1][
123
      j ] > time[i]):
                            if (startEndLow[0][j] > arrivalTimesLow[j]):
                                length += 1
125
                   lowPriority.append(length)
               allLengthsLowPriority[k] = deepcopy(lowPriority)
               for i in range (0, len(time)):
                   length = 0
131
                    for j in range(0, len(arrivalTimesHigh)):
```

```
if (time[i] >= arrivalTimesHigh[j] and startEndHigh
133
      [1][j] > time[i]:
                            if (startEndHigh[0][j] > arrivalTimesHigh[j]):
                                length += 1
135
                   high Priority.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"
           print ("Average inter-arrival time -- (HIGH) ---", round ((sum(
143
      totalInterArrivalHigh))/(len(totalInterArrivalHigh)),4), "us"
           print("Average inter-arrival time -- (TOTAL) --", round((sum(
      totalInterArrivalHigh) + sum(totalInterArrivalLow))/(len(
      totalInterArrivalHigh) + len(totalInterArrivalLow)), 4), "us")
           print("Average transmission time --- (LOW) ----", round(sum(
145
      totalTransmissionLow)/len(totalTransmissionLow), 4), "us")
           print ("Average transmission time --- (HIGH) ---", round (sum (
      totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us")
           print("Average transmission time --- (TOTAL) --", round((sum(
147
      totalTransmissionHigh) + sum(totalTransmissionLow))/(len(
      totalTransmissionHigh)+len(totalTransmissionLow)), 4), "us")
           print ("Average response time ----- (LOW) ---", round (sum (
      totalResponseLow)/len(totalResponseLow), 4), "us")
           print ("Average response time ----- (HIGH) ---", round (sum (
149
      totalResponseHigh)/len(totalResponseHigh), 4), "us")
print("Average response time ----- (TOTAL) --", round((sum(
      totalResponseLow) + sum(totalResponseHigh))/(len(totalResponseLow) +
      len(totalResponseHigh)), 4) , "us" )
           print("The average packet size ---- (LOW) ----", round(sum(
151
      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")
           print ("Average delays ----- (HIGH) ---", round (sum (
155
      totalAverageDelaysHigh)/len(totalAverageDelaysHigh), 4), "us")
           print("Average delays ----- (TOTAL) --", round((sum(
      totalAverageDelaysLow) + sum(totalAverageDelaysHigh))/(len(
      totalAverageDelaysHigh) + len(totalAverageDelaysLow)), 4), "us")
           print ("Average arrival rate (lambda) (LOW) ----", round ((len (
157
      totalAveragePacketSizeLow))/((sum(totalInterArrivalLow)))*10**6, 4), "
      packets/second")
           print ("Average arrival rate (lambda) (HIGH) ---", round ((len (
      totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh)))*10**6, 4),
       "packets/second")
```

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

```
plt.legend(loc='best')
      plt.xlabel('Time (seconds)')
199
      plt.ylabel ('Customers waiting in queue')
      plt.title('Non-Preemptive Varying arrRate Low LowPriority')
201
      plt.show()
      plt.figure()
      for t in range(0, len(allQueuesHigh)):
205
          plt.plot(time, allQueuesHigh[t], label="High" + str(arrRates[t])
      + " us")
      plt.legend(loc='best')
207
      plt.xlabel('Time (seconds)')
      plt.ylabel('Customers waiting in queue')
      plt.title('Non-Preemptive Varying arrRate High Priority')
      plt.show()
211
  if( doVaryingArrivalPreempt == True):
     Averaged Preemptive varying arrRate
215
     ######")
      print ("
     217
      allQueuesLow = []
      allQueuesHigh = []
219
      allQueuesComb = []
      for 1 in range (0, len(arrRates)):
          time = np.arange(0, 125*10**6, 1000000)
          totalInterArrivalLow
          totalInterArrivalHigh
227
          totalTransmissionLow
          totalTransmissionHigh
          totalResponseLow
          totalResponseHigh
          total Average Packet Size Low\\
          totalAveragePacketSizeHigh =
          totalAverageDelaysLow
          totalAverageDelaysHigh
          totalArrivalRateLow
          totalArrivalRateHigh
          total Service Rate Low\\
          totalServiceRateHigh
239
          allCombinedLengths
                                          * averagedOver
                                   =
          allLengthsHighPriority
                                          * averagedOver
          allLengthsLowPriority
                                          * averagedOver
243
          for k in range (0, averagedOver):
              packets = [[],[]]
245
             transCap = linkCapacity
247
             # Read the csv file data into respective lists to store the
     data
```

```
interArrivalHigh = list(np.random.exponential(arrRates[1]),
249
      int(100/(arrRates[1]*10**-6))))
                                = list (np.random.exponential(1000, int(100/(
               packetSizeHigh
      arrRates[1]*10**-6))))
               interArrivalLow = list (np.random.exponential (2000, 50000))
                                = list(np.random.exponential(1000, 50000))
               packetSizeLow
253
               # Create the list of low priority packets
               for i in range(0, len(interArrivalLow)):
                   lowPacket = Packet(packetSizeLow[i], interArrivalLow[i],
      'LOW')
                   packets [0]. append (lowPacket)
257
               # Create the list of high priority packets
259
               for i in range(0, len(interArrivalHigh)):
                   highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i]
261
        'HIGH')
                   packets [1]. append (highPacket)
263
               # Get the parameters
               arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
265
      allPackets = calcPreemptive(transCap, packets)
               sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))),
      float (sum(packetSizeHigh))/len(packetSizeHigh)]
               if (k == 0):
                   print ("Link capacity -----",
269
      transCap / 10 * * 6, "Mbps")
                   print ("Number of low priority packets ----", len (
      packetSizeLow), "packets")
                   print ("Number of high priority packets ----", len (
271
      packetSizeHigh), "packets")
                   print ("Total number of packets ----", len (
      packetSizeLow) + len(packetSizeHigh), "packets" )
273
               print(1,k)
               total Inter Arrival Low\\
                                           += interArrivalLow
275
               totalInterArrivalHigh
                                           += interArrivalHigh
               totalTransmissionLow
                                           += transTimes [0]
               totalTransmissionHigh
                                           += transTimes [1]
               totalResponseLow
                                           += respTimes[0]
               totalResponseHigh
                                           += respTimes[1]
                                          += packetSizeLow
               totalAveragePacketSizeLow
               totalAveragePacketSizeHigh += packetSizeHigh
               totalAverageDelaysLow
                                          += queuingDelays [0]
283
                                           += queuingDelays[1]
               totalAverageDelaysHigh
285
               startEnd = [[],[]]
               arrivalTimes = []
287
               startEndLow = [[],[]]
               arrivalTimesLow = []
               startEndHigh = [[],[]]
               arrivalTimesHigh =
291
               for i in allPackets:
                   startEnd[0].append(i.startTime)
```

```
startEnd[1].append(i.endTime)
295
                    arrivalTimes.append(i.arrivalTime)
297
                    if (i.priority == 'LOW'):
                        startEndLow[0].append(i.startTime)
                        startEndLow[1].append(i.endTime)
                        arrivalTimesLow.append(i.arrivalTime)
301
                    if (i.priority = 'HIGH'):
                        startEndHigh [0].append(i.startTime)
                        startEndHigh [1].append(i.endTime)
305
                        arrivalTimesHigh.append(i.arrivalTime)
               combined = []
               lowPriority = []
309
               high Priority = []
               for i in range (0, len(time)):
                    length = 0
313
                    for j in range(0, len(arrivalTimesLow)):
                        if (time[i] >= arrivalTimesLow[j] and startEndLow[1][
315
      j > time[i]:
                            if \ (startEndLow \, [\, 0\, ] \, [\, j\, ] \ > \ arrivalTimesLow \, [\, j\, ]\, ):
                                length += 1
317
                    lowPriority.append(length)
319
               allLengthsLowPriority[k] = deepcopy(lowPriority)
               for i in range (0, len(time)):
                    length = 0
                    for j in range(0, len(arrivalTimesHigh)):
                        if (time[i] >= arrivalTimesHigh[j] and startEndHigh
      [1][j] > time[i]:
                             if (startEndHigh[0][j] > arrivalTimesHigh[j]):
                                 length += 1
327
                    high Priority.append (length)
329
               allLengthsHighPriority[k] = deepcopy(highPriority)
               allCombinedLengths[k] = list( map(add, highPriority,
331
      lowPriority))
333
           print ("Average inter-arrival time -- (LOW) ----", round ((sum(
      totalInterArrivalLow))/(len(totalInterArrivalLow)), 4), "us"
           print ("Average inter-arrival time -- (HIGH) ---", round ((sum(
335
      totalInterArrivalHigh))/(len(totalInterArrivalHigh)),4), "us")
           print("Average inter-arrival time -- (TOTAL) --", round((sum(
      totalInterArrivalHigh) + sum(totalInterArrivalLow))/(len(
      totalInterArrivalHigh) + len(totalInterArrivalLow)), 4), "us")
           print ("Average transmission time --- (LOW) ----", round (sum (
337
      total Transmission Low)/len\left(total Transmission Low\right),~4)~,~"us"~)
           print ("Average transmission time --- (HIGH) ---", round (sum (
      totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us")
           print("Average transmission time --- (TOTAL) --", round((sum(
339
      totalTransmissionHigh) + sum(totalTransmissionLow))/(len(
      totalTransmissionHigh)+len(totalTransmissionLow)), 4), "us")
```

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

```
367
           averagedCombinedLength = np.asarray(averagedCombinedLength)
           averagedLowPriorLength = np.asarray(averagedLowPriorLength)
369
           averagedHighPrioLength = np.asarray(averagedHighPrioLength)
           for x in range (0, len (averagedCombinedLength)):
               averagedCombinedLength[x] = averagedCombinedLength[x]/
      averagedOver
               averagedLowPriorLength[x] = averagedLowPriorLength[x]/
375
      averagedOver
               averagedHighPrioLength[x] = averagedHighPrioLength[x]/
      averagedOver
377
           allQueuesComb.append(deepcopy(averagedCombinedLength))
           allQueuesHigh.append(deepcopy(averagedHighPrioLength))
379
           allQueuesLow.append(deepcopy(averagedLowPriorLength))
381
      averagedQueuesHigh = [0]*len(allQueuesHigh[0])
      for a in range (0, len(allQueuesLow)):
             print ("Average queue length low priority" + str (arrRates [a]),
385
      round \left(sum \left(all Queues Low \left[a\right]\right) / len \left(all Queues Low \left[a\right]\right), \ 4\right), \ "packets / second"\right)
             print("Average queue length high priority " + str(arrRates[a])
      , round (sum (all Queues High [a]) / len (all Queues High [a]), 4), "packets/
      second")
387
      plt.figure()
       for t in range (0, len(allQueuesHigh)):
389
           plt.plot(time, allQueuesLow[t], label="Low" + str(arrRates[t]) +
         us")
391
      plt.legend(loc='best')
       plt.xlabel('Time (seconds)')
393
      plt.ylabel('Customers waiting in queue')
       plt.title('Preemptive Varying arrRate Low LowPriority')
395
      plt.show()
397
       plt.figure()
       for t in range (0, len(allQueuesHigh)):
           plt.plot(time, allQueuesHigh[t], label="High" + str(arrRates[t])
       + " us")
       plt.legend(loc='best')
401
      plt.xlabel('Time (seconds)')
       plt.ylabel ('Customers waiting in queue')
403
       plt.title('Preemptive Varying arrRate High Priority')
      plt.show()
405
  if ( doVaryingArrivalNoPriority == True):
      print ("
407
     <del>#################################</del>*)
      Averaged No Priority varying arrRate
      <del>######</del>" )
      print ('
409
     411
      allQueuesLow = []
```

```
allQueuesHigh = []
       allQueuesComb = []
413
       for 1 in range (0, len(arrRates)):
415
           time = np.arange(0, 101*10**6, 1000000)
           totalInterArrivalLow
410
           totalInterArrivalHigh
           totalTransmissionLow
421
           totalTransmissionHigh
           totalResponseLow
423
           totalResponseHigh
           total Average Packet Size Low
425
           totalAveragePacketSizeHigh =
           totalAverageDelaysLow
427
           totalAverageDelaysHigh
           totalArrivalRateLow
429
           totalArrivalRateHigh
           total Service Rate Low\\
431
           totalServiceRateHigh
           allCombinedLengths
                                                * averagedOver
433
           allLengthsHighPriority
                                        =
                                                * averagedOver
           all Lengths Low Priority\\
                                                * averagedOver
           for k in range (0, averaged Over):
437
               packets = [[],[]]
               transCap = linkCapacity
439
               # Read the csv file data into respective lists to store the
441
      data
               interArrivalHigh = list (np.random.exponential (arrRates [1]),
      int (100/(arrRates[1]*10**-6)))
               packetSizeHigh
                                 = list (np.random.exponential (1000, int (100/(
443
      arrRates[1]*10**-6))))
               interArrivalLow = list (np.random.exponential (2000, 50000))
               packetSizeLow
                                  = list (np.random.exponential (1000, 50000))
445
               # Create the list of low priority packets
447
                for i in range (0, len (interArrivalLow)):
                    lowPacket = Packet(packetSizeLow[i], interArrivalLow[i],
449
      'LOW')
                    packets [0]. append (lowPacket)
               # Create the list of high priority packets
                for i in range(0, len(interArrivalHigh)):
453
                    highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i]
         'HIGH')
                    packets [1]. append (highPacket)
455
               # Get the parameters
               arrivalTimes, transTimes, queuingDelays, startEnd, respTimes,
      allPackets = calcNoPriority(transCap, packets)
               sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))),
459
      float (sum(packetSizeHigh))/len(packetSizeHigh)]
```

```
if (k == 0):
461
                   print ("Link capacity ----",
      transCap / 10 * * 6, "Mbps")
                    print ("Number of low priority packets ----", len (
463
      packetSizeLow), "packets")
                   print ("Number of high priority packets ----", len (
      packetSizeHigh), "packets")
                   print ("Total number of packets ----", len (
465
      packetSizeLow) + len(packetSizeHigh), "packets")
               print(1,k)
467
               totalInterArrivalLow
                                           += interArrivalLow
               totalInterArrivalHigh
                                           += interArrivalHigh
               total Transmission Low\\
                                           += transTimes [0]
               totalTransmissionHigh
                                           += transTimes[1]
471
               totalResponseLow
                                           += respTimes [0]
               totalResponseHigh
                                           += respTimes[1]
               totalAveragePacketSizeLow
                                           += packetSizeLow
               totalAveragePacketSizeHigh += packetSizeHigh
475
               totalAverageDelaysLow
                                           += queuingDelays [0]
               totalAverageDelaysHigh
                                           += queuingDelays [1]
               startEnd = [[],[]]
479
               arrivalTimes = []
               startEndLow = [[],[]]
               arrivalTimesLow = []
               startEndHigh = [[],[]]
483
               arrivalTimesHigh = []
               for i in allPackets:
                   startEnd[0].append(i.startTime)
487
                   startEnd[1].append(i.endTime)
                   arrivalTimes.append(i.arrivalTime)
489
                   if (i.priority = 'LOW'):
491
                       startEndLow[0].append(i.startTime)
                       startEndLow[1].append(i.endTime)
493
                       arrivalTimesLow.append(i.arrivalTime)
495
                   if (i.priority = 'HIGH'):
                       startEndHigh[0].append(i.startTime)
497
                       startEndHigh[1].append(i.endTime)
                       arrivalTimesHigh.append(i.arrivalTime)
499
               combined = []
501
               lowPriority = []
               high Priority = []
503
505
               for i in range (0, len(time)):
                   length = 0
                   for j in range (0, len (arrivalTimesLow)):
                        if (time[i] >= arrivalTimesLow[j] and startEndLow[1][
      j > time[i]:
                            if (startEndLow[0][j] > arrivalTimesLow[j]):
511
                                length += 1
```

```
lowPriority.append(length)
               allLengthsLowPriority[k] = deepcopy(lowPriority)
               for i in range (0, len(time)):
                   length = 0
                   for j in range (0, len (arrivalTimesHigh)):
                       if (time[i] >= arrivalTimesHigh[j] and startEndHigh
      [1][j] > time[i]:
                           if (\text{startEndHigh}[0][j] > \text{arrivalTimesHigh}[j]):
                               length += 1
                   high Priority.append(length)
               allLengthsHighPriority[k] = deepcopy(highPriority)
              allCombinedLengths[k] = list(map(add, highPriority,
      lowPriority))
527
          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")
           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")
          print ("Average transmission time --- (HIGH) ---", round (sum (
      totalTransmissionHigh)/len(totalTransmissionHigh), 4), "us"
          print ("Average transmission time --- (TOTAL) --", round ((sum (
533
      totalTransmissionHigh) + sum(totalTransmissionLow))/(len(
      totalTransmissionHigh)+len(totalTransmissionLow)), 4), "us")
          print ("Average response time ----- (LOW) ---", round (sum (
      totalResponseLow)/len(totalResponseLow), 4), "us")
          print("Average response time ----- (HIGH) ---", round(sum(
535
      totalResponseHigh)/len(totalResponseHigh), 4), "us")
           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"
          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")
           print ("Average delays ----- (HIGH) ---", round (sum (
541
      totalAverageDelaysHigh)/len(totalAverageDelaysHigh), 4), "us"
           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")
           print("Average arrival rate (lambda) (HIGH) ---", round((len(
      totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh)))*10**6, 4),
       "packets/second")
           print("Average arrival rate (lambda) (TOTAL) --", round((len(
545
      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")
           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
           averagedCombinedLength = [0] * len(allCombinedLengths[0])
           averagedLowPriorLength = [0] * len(allLengthsLowPriority[0])
           averagedHighPrioLength = [0] * len(allLengthsHighPriority[0])
           for x in range(0, len(allLengthsHighPriority)):
                for y in range (0, len (allLengthsHighPriority[x])):
                    averagedCombinedLength[y] += allCombinedLengths[x][y]
                    averagedLowPriorLength[y] += allLengthsLowPriority[x][y]
                    averagedHighPrioLength[y] += allLengthsHighPriority[x][y]
561
           averagedCombinedLength = np.asarray(averagedCombinedLength)
           averagedLowPriorLength = np.asarray(averagedLowPriorLength)
563
           averagedHighPrioLength = np.asarray(averagedHighPrioLength)
565
           for x in range (0, len (averagedCombinedLength)):
567
               averagedCombinedLength[x] = averagedCombinedLength[x]/
      averagedOver
               averagedLowPriorLength[x] = averagedLowPriorLength[x]/
569
      averagedOver
               averagedHighPrioLength[x] = averagedHighPrioLength[x]/
      averagedOver
           allQueuesComb.append(deepcopy(averagedCombinedLength))
           allQueuesHigh.append(deepcopy(averagedHighPrioLength))
573
           allQueuesLow.append(deepcopy(averagedLowPriorLength))
       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 (all Queues High [a]) / len (all Queues High [a]), 4), "packets/
      second")
```

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

Code/DriverVary.py