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

ERP 420

Research Project

Practical 2: Priority Queuing

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

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

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

ELO 4: Investigations, experiments and data analysis

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

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

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

Friday 20th September, 2019

1 Introduction

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

2 Background

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

2.1 Preemptive Non-resume Priority Scheduling

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

2.2 Non-preemptive Priority Scheduling

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

2.3 M/M/1 Performance Parameter Notations

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

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

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

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

L_q – Average number of customers waiting in line.

3 Design and Implementation

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

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

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

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

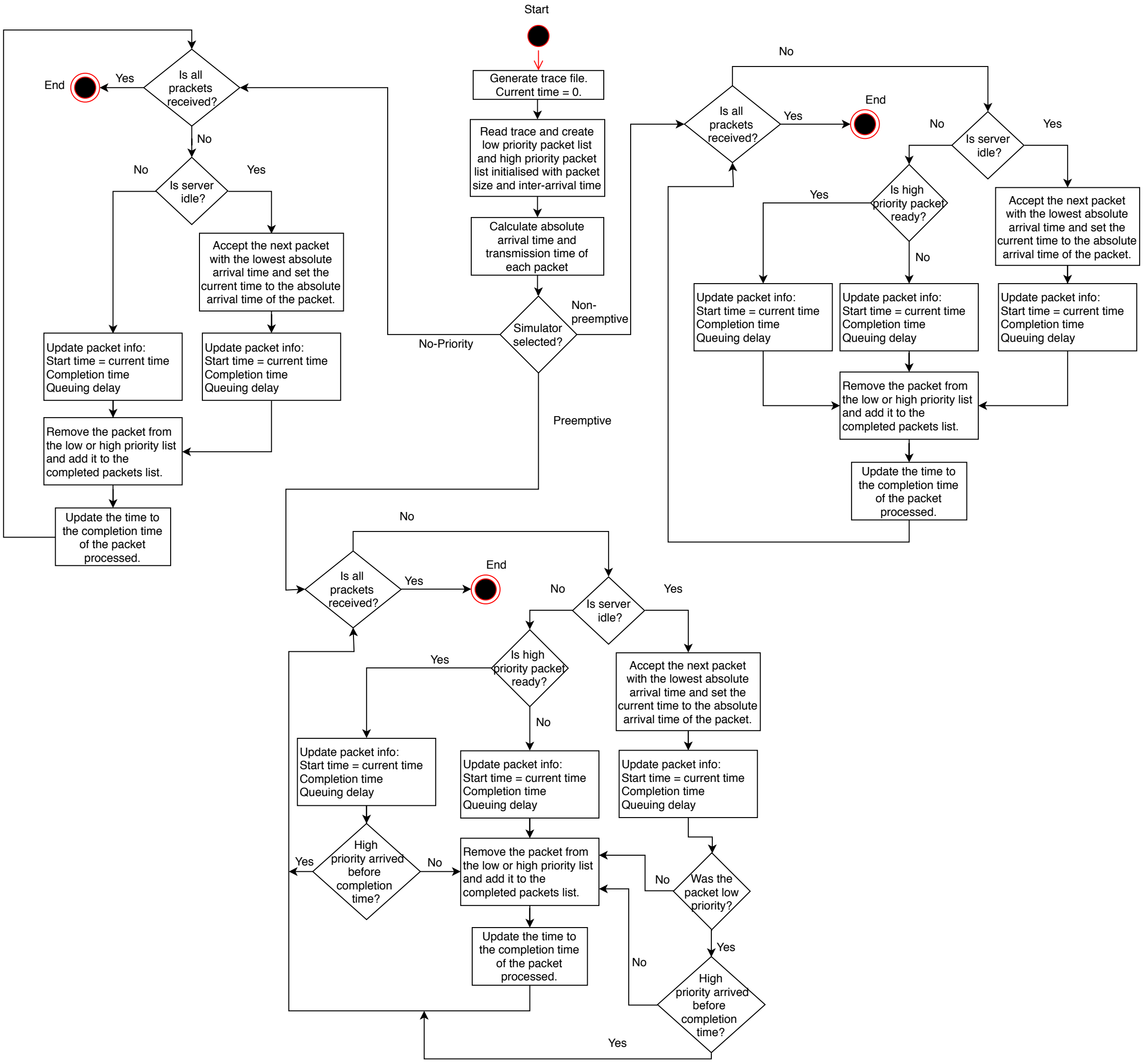

Start
Generate trace file. Current time = 0.
Read trace and create low priority packet list and high priority packet list initialised with packet size and inter-arrival time
Calculate absolute arrival time and transmission time of each packet
Simulator selected?
No-Priority
Non-preemptive
Preemptive
End
Yes
Is all prackets received?
No
No
Is server idle?
Yes
Accept the next packet with the lowest absolute arrival time and set the current time to the absolute arrival time of the packet.
Update packet info: Start time = current time Completion time Queuing delay
Update packet info: Start time = current time Completion time Queuing delay
Remove the packet from the low or high priority list and add it to the completed packets list.
Update the time to the completion time of the packet processed.
No
Is all packets received?
Yes
End
No
Is server idle?
Yes
Yes
Is high priority packet ready?
No
Accept the next packet with the lowest absolute arrival time and set the current time to the absolute arrival time of the packet.
Update packet info: Start time = current time Completion time Queuing delay
Update packet info: Start time = current time Completion time Queuing delay
Update packet info: Start time = current time Completion time Queuing delay
Remove the packet from the low or high priority list and add it to the completed packets list.
Update the time to the completion time of the packet processed.
No
Is all packets received?
Yes
End
No
Is server idle?
Yes
Yes
Is high priority packet ready?
No
Accept the next packet with the lowest absolute arrival time and set the current time to the absolute arrival time of the packet.
Update packet info: Start time = current time Completion time Queuing delay
Update packet info: Start time = current time Completion time Queuing delay
Update packet info: Start time = current time Completion time Queuing delay
Yes
High priority arrived before completion time?
No
Remove the packet from the low or high priority list and add it to the completed packets list.
No
Was the packet low priority?
Yes
No
High priority arrived before completion time?
Yes
Update the time to the completion time of the packet processed.

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

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

$$\text{Transmission time} = \frac{\text{Packet size}}{\text{Link capacity}}. \tag{3.1}$$

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

3.1 Non-preemptive Simulator

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

3.2 Preemptive Non-resume Simulator

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

3.3 No Priority Simulator

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

3.4 Queue Length

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

3.5 Simulations

3.5.1 Given Trace Files

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

3.5.2 Averaged Results

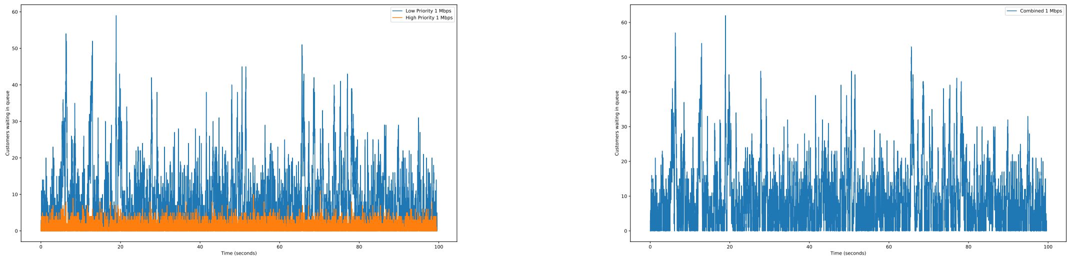
For this simulation, the same performance parameters will be calculated as in Section 3.5.1 and the queue length will also be plotted against time. The results will, however, be averaged out over 150 runs. The simulator will, therefore, generate the traces file 150 times, process the data and average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

3.5.3 Varying Arrival Rates

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

4 Results

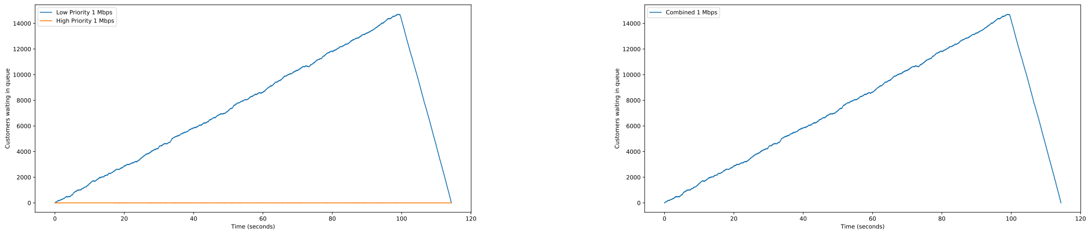
The first results that are provided will be for the given trace files for high and low priority packets. Table 2 shows the calculated performance parameters associated with each one of the simulators. The performance parameters were calculated for the high priority packets separately from the low priority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



(a) High and low priority queues.

(b) Combined priorities.

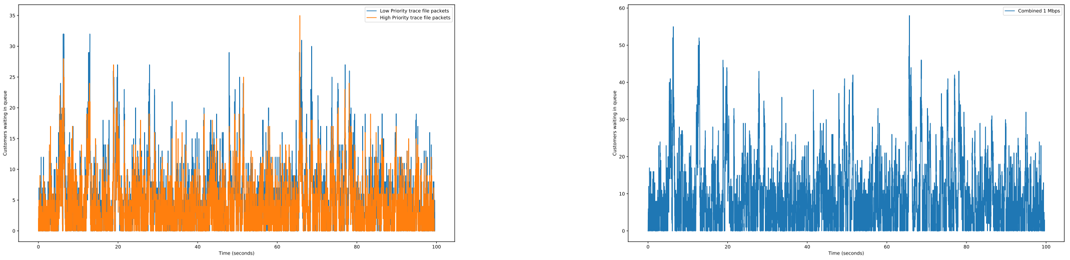
Figure 2. High and low priority queue lengths plotted against time separately 2(a) and combined 2(b) for a 1 Mbps non-preemptive priority simulator.



(a) High and low priority queues.

(b) Combined priorities.

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



(a) Separate file queue lengths.

(b) Combined file queue lengths.

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

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

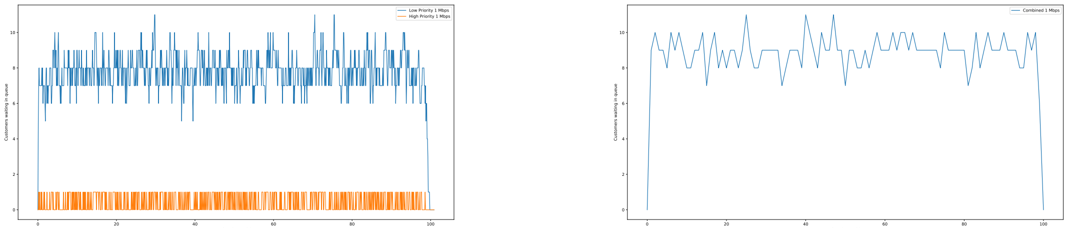
		Calculated for packet priority			
Simulator	**Parameter**	**Low**	**High**	**Combined**	**Unit**
Non-preemptive	Average response time	16340.423	2517.5917	10196.9424	μs
Preemptive		16727115.2507	1673.0544	9293585.3857	μs
No priority		10159.2036	10132.7128	10147.4299	μs
Non-preemptive	Average queueing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

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

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

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

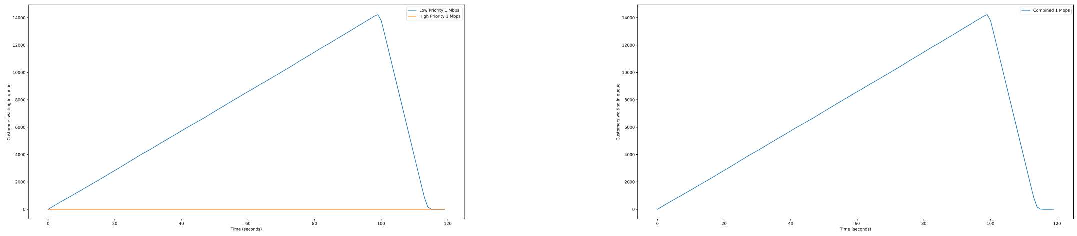
Table 3. Performance parameters calculated for the given trace file.



(a) High and low priority queues.

(b) Combined priorities.

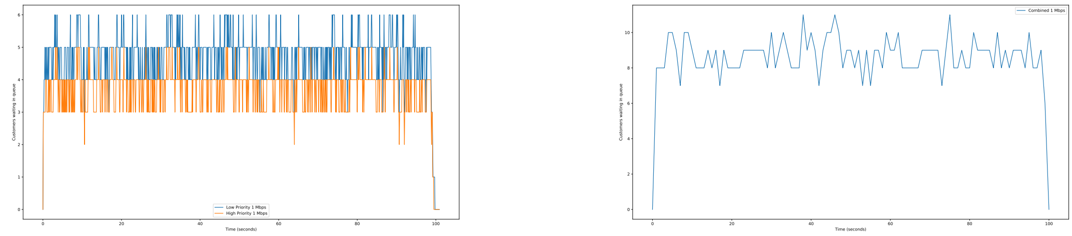
Figure 5. High and low priority queue lengths plotted against time separately 5(a) and combined 5(b) for a 1 Mbps non-preemptive priority simulator, averaged over 150 simulations.



(a) High and low priority queues.

(b) Combined priorities.

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



(a) Separate file queue lengths.

(b) Combined file queue lengths.

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

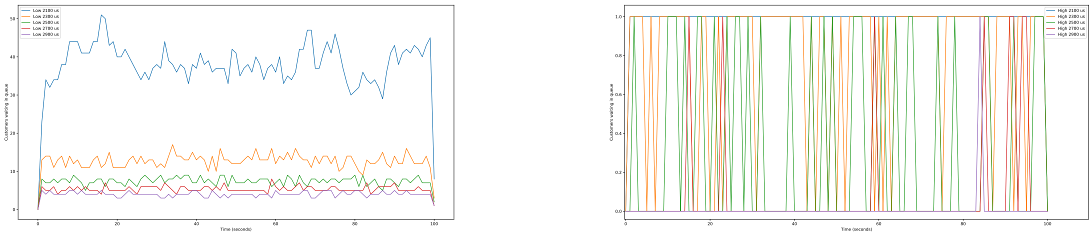
		Calculated for packet priority			
Simulator	**Parameter**	**Low**	**High**	**Combined**	**Unit**
Non-preemptive	Average response time	16807.1566	2511.3844	10453.48	μs
Preemptive		16520435.8291	1672.9732	9178763.4487	μs
No priority		10368.7603	10387.9942	10377.3087	μs
Non-preemptive	Average queueing delays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

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

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

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

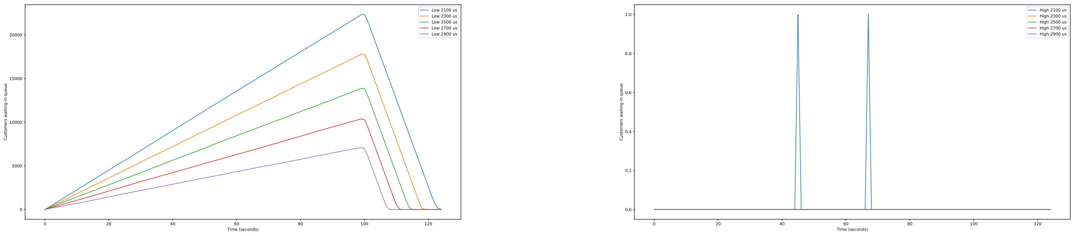
Table 5. Averaged performance parameters over 100 simulations, calculated for a 1 Mbps link capacity with varying the average inter-arrival times (IAT) in microseconds.



(a) Low priority queue length.

(b) High priority queue length.

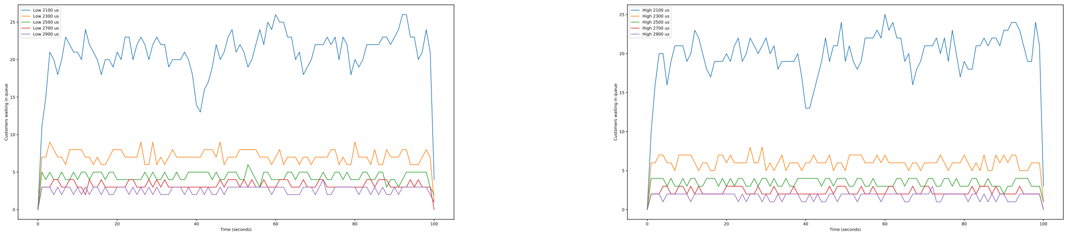
Figure 8. Low 8(a) and high 8(b) priority queue lengths plotted against time for varying mean inter-arrival time of high priority packets for a 1 Mbps non-preemptive priority simulator, averaged over 100 simulations.



(a) Low priority queue length.

(b) High priority queue length.

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



(a) First file queue length.

(b) Second queue length.

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

5 Discussion

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

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

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

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

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

6 Conclusion

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

7 References

[1] L. Strydom, *ERP 420 – Practical 2 Priority Queuing and Queuing Management Systems v1.1*, University of Pretoria, 8 Sept. 2019.

[2] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains*, 2nd ed. Wiley, 2006.

A Simulators Python Souce Code

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

@author: project
"""

import csv
import time


class Packet:
    def __init__(self, size, interArrivalTime, priority):
        self.size             = size
        self.interArrivalTime = interArrivalTime
        self.priority         = priority
        self.arrivalTime      = 0
        self.transTime        = 0
        self.queueDelay       = 0
        self.startTime        = 0
        self.endTime          = 0
        self.responseTime     = 0

# Read the text file and the inter-arrival time and the size
def readCSV(filePath):
    arrival = []
    size = []

    with open(filePath,'r') as f:
        data = csv.reader(f)
        for row in data:
            arrival.append(row[0])
            size.append(row[1])

    # Convert the strings lists to integer lists
    arrival = list(map(int, arrival))
    size = list(map(int, size))

    return arrival, size


def getTraceInfo():
    # Read the csv file data into respective lists to store the data
    interArrivalHigh, packetSizeHigh = readCSV('HighPriority.txt')
    interArrivalLow , packetSizeLow  = readCSV('LowPriority.txt')
    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(
   interArrivalHigh))
    print("Avg size Low          :", sum(packetSizeLow)/len(packetSizeLow
   ))
```

```python
    print("Avg size High         :", sum(packetSizeHigh)/len(packetSizeHigh))
    print("Total interarrival    :", (sum(interArrivalHigh)+sum(interArrivalLow))/(len(interArrivalHigh)+len(interArrivalLow)))


getTraceInfo()

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

    # Calculate when the packets were received for low priority
    for i in range(0, len(packets[0])):
        if (i == 0):
            packets[0][i].arrivalTime = packets[0][i].interArrivalTime

        else:
            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 + 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

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

        toProcess = None

        # If a high priority packet was received
        if (len(packets[1]) > 0 and packets[1][0].arrivalTime <= currTime):
            toProcess = packets[1][0]
            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 <= 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:
            # If there are still packets to be processed of high and low priority
            if (len(packets[1]) > 0 and len(packets[0]) > 0):
                if (packets[1][0].arrivalTime <= 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]
                currTime = packets[1][0].arrivalTime
                packets[1] = packets[1][1:]

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

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

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

    arrivalTimes   = [[],[]]
    transTimes     = [[],[]]
    queuingDelays  = [[],[]]
    startEnd       = [[],[]]
    respTimes      = [[],[]]


    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)
            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 transmission
# capacity as an integer and returns arrivalTimes, transTimes, queuingDelays,
# startEnd, idleTimes, respTimes.
def calcPreemptive(transCap, packets):
    currTime         = 0
    completedPackets = []

    # Calculate when the packets were received for low priority
    for i in range(0, len(packets[0])):
        if (i == 0):
            packets[0][i].arrivalTime = packets[0][i].interArrivalTime

        else:
            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 + 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

    # 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 or len(packets[1]) > 0):
        # For the first packet arriving
        toProcess = None

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

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

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

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

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

            # 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

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

            completedPackets.append(toProcess)

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

    arrivalTimes   = [[],[]]
    transTimes     = [[],[]]
    queuingDelays  = [[],[]]
    startEnd       = [[],[]]
    respTimes      = [[],[]]


    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)
            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 correnspending to
# [[class:packetLowPriority], class[packetHighPriority]] and the transmission
# capacity as an integer and returns arrivalTimes, transTimes, queuingDelays,
# startEnd, idleTimes, respTimes.
def calcNoPriority(transCap, packets):

    currTime         = 0
    completedPackets = []

    # Calculate when the packets were received for low priority
    for i in range(0, len(packets[0])):
        if (i == 0):
            packets[0][i].arrivalTime = packets[0][i].interArrivalTime

        else:
            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 +
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

# 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 or len(packets[1]) > 0):
    # For the first packet arriving
    toProcess = None

    if (len(packets[1]) > 0 and len(packets[0]) > 0 and packets
[1][0].arrivalTime <= currTime and packets[0][0].arrivalTime <=
currTime):
        if (packets[1][0].arrivalTime <= packets[0][0].arrivalTime):
            toProcess = packets[1][0]
            packets[1] = packets[1][1:]

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

    # If a high priority packet was received
    elif (len(packets[1]) > 0 and packets[1][0].arrivalTime <=
currTime):
        toProcess = packets[1][0]
        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 <=
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:
        # If there are still packets to be processed of high and low
priority
        if (len(packets[1]) > 0 and len(packets[0]) > 0):
            if (packets[1][0].arrivalTime < 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]
            currTime = packets[1][0].arrivalTime
            packets[1] = packets[1][1:]

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

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

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

arrivalTimes   = [[],[]]
transTimes     = [[],[]]
queuingDelays  = [[],[]]
startEnd       = [[],[]]
respTimes      = [[],[]]


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)
        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
```

Code/PriorityQueuing.py

B Trace File Driver

```
# -*- coding: utf-8 -*-
"""
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
import time as t
from operator import add

doNonPreemptiveTrace        = True
doPreemptiveTrace           = False
doNoPriority                = False


linkCapacity = 1*10**6


if (doNoPriority == True):
    print("
    ##################################################################################")
    print("########          Processing no priority queuing
    #######")
    print("
    ##################################################################################")

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

    transCap = linkCapacity

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

    # 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
    for i in range(0, len(interArrivalHigh)):
        highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH')
        packets[1].append(highPacket)

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

    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 ----------------", 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(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(transTimes[1]) + sum(transTimes[0]))/(len(transTimes[1])+len(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[1])/len(respTimes[1]), 4), "us" )
    print("Average response time ------- (TOTAL) --", round((sum(respTimes[0]) + sum(respTimes[1]))/(len(respTimes[0]) + len(respTimes[1])), 4) , "us" )
    print("The average packet size ----- (LOW) ----", round(sizeAvg[0], 4) , "bits" )
    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(queuingDelays[1])/len(queuingDelays[1]), 4), "us")
    print("Average delays -------------- (TOTAL) --", round((sum(queuingDelays[0]) + sum(queuingDelays[1]))/(len(queuingDelays[0]) + len(queuingDelays[1])), 4), "us")
    print("Average arrival rate (lambda) (LOW) ----", round((len(packetSizeLow))/((sum(interArrivalLow)))*10**6, 4), "packets/second")
```

```python
    print("Average arrival rate (lambda) (HIGH) ---", round((len(packetSizeHigh))/((sum(interArrivalHigh)))*10**6, 4), "packets/second")
    print("Average arrival rate (lambda) (TOTAL) --", round((len(packetSizeLow) + len(packetSizeHigh))/(((sum(interArrivalHigh)*40000 + sum(interArrivalLow)*50000)/90000))*10**6, 4), "packets/second")
    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/(((sizeAvg[0]*50000 + sizeAvg[1]*40000)/90000)), 4), "packets/second")

    startEnd = [[],[]]
    arrivalTimes = []
    startEndLow = [[],[]]
    arrivalTimesLow = []
    startEndHigh = [[],[]]
    arrivalTimesHigh = []

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

        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)

    time = np.arange(0, startEnd[1][-1], 10000)
    combined = []
    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]):
                    length += 1
        lowPriority.append(length)

    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
        highPriority.append(length)

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

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


    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('No priority combined Trace')
    plt.show()

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


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

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

    transCap = linkCapacity

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

    # 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
    for i in range(0, len(interArrivalHigh)):
        highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH')
        packets[1].append(highPacket)

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

    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 ----------------", 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(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(transTimes[1]) + sum(transTimes[0]))/(len(transTimes[1])+len(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[1])/len(respTimes[1]), 4), "us" )
    print("Average response time ------- (TOTAL) --", round((sum(respTimes[0]) + sum(respTimes[1]))/(len(respTimes[0]) + len(respTimes[1])), 4) , "us" )
    print("The average packet size ----- (LOW) ----", round(sizeAvg[0], 4) , "bits" )
    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(queuingDelays[1])/len(queuingDelays[1]), 4), "us")
    print("Average delays -------------- (TOTAL) --", round((sum(queuingDelays[0]) + sum(queuingDelays[1]))/(len(queuingDelays[0]) + len(queuingDelays[1])), 4), "us")
    print("Average arrival rate (lambda) (LOW) ----", round((len(
packetSizeLow))/((sum(interArrivalLow)))*10**6, 4), "packets/second")
    print("Average arrival rate (lambda) (HIGH) ---", round((len(
packetSizeHigh))/((sum(interArrivalHigh)))*10**6, 4), "packets/second"
)
    print("Average arrival rate (lambda) (TOTAL) --", round((len(
packetSizeLow) + len(packetSizeHigh))/(((sum(interArrivalHigh)*40000 +
 sum(interArrivalLow)*50000)/90000))*10**6, 4), "packets/second")
    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/(((
sizeAvg[0]*50000 + sizeAvg[1]*40000)/90000)), 4), "packets/second")


    startEnd = [[],[]]
    arrivalTimes = []
    startEndLow = [[],[]]
    arrivalTimesLow = []
    startEndHigh = [[],[]]
    arrivalTimesHigh = []

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

        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)

    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] >
time[i]):
                if (startEndLow[0][j] > arrivalTimesLow[j]):
                    length += 1
        lowPriority.append(length)

    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
            highPriority.append(length)

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


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


    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')
    plt.show()

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


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

    transCap = linkCapacity

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

    # 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
    for i in range(0, len(interArrivalHigh)):
        highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH')
        packets[1].append(highPacket)

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

    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 ----------------", 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(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(transTimes[1]) + sum(transTimes[0]))/(len(transTimes[1])+len(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[1])/len(respTimes[1]), 4), "us" )
    print("Average response time ------- (TOTAL) --", round((sum(respTimes[0]) + sum(respTimes[1]))/(len(respTimes[0]) + len(respTimes[1])), 4) , "us" )
    print("The average packet size ----- (LOW) ----", round(sizeAvg[0], 4) , "bits" )
    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(queuingDelays[1])/len(queuingDelays[1]), 4), "us")
    print("Average delays -------------- (TOTAL) --", round((sum(queuingDelays[0]) + sum(queuingDelays[1]))/(len(queuingDelays[0]) +
len(queuingDelays[1])), 4), "us")
    print("Average arrival rate (lambda) (LOW) ----", round((len(
packetSizeLow))/((sum(interArrivalLow)))*10**6, 4), "packets/second")
    print("Average arrival rate (lambda) (HIGH) ---", round((len(
packetSizeHigh))/((sum(interArrivalHigh)))*10**6, 4), "packets/second"
)
    print("Average arrival rate (lambda) (TOTAL) --", round((len(
packetSizeLow) + len(packetSizeHigh))/(((sum(interArrivalHigh)*40000 +
 sum(interArrivalLow)*50000)/90000))*10**6, 4), "packets/second")
    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/(((
sizeAvg[0]*50000 + sizeAvg[1]*40000)/90000)), 4), "packets/second")


    startEnd = [[],[]]
    arrivalTimes = []
    startEndLow = [[],[]]
    arrivalTimesLow = []
    startEndHigh = [[],[]]
    arrivalTimesHigh = []

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

        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)

    time = np.arange(0, 120*10**6, 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[0][j] >
time[i] and startEndLow[0][j] > arrivalTimesLow[j]):
                length += 1
        lowPriority.append(length)

    for i in range(0, len(time)):
        length = 0
        for j in range(0, len(arrivalTimesHigh)):
            if (time[i] >= arrivalTimesHigh[j] and startEndHigh[0][j] > time[i] and startEndHigh[0][j] > arrivalTimesHigh[j]):
                    length += 1
        highPriority.append(length)

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

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


    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('Combined Preemptive queues Trace')
    plt.show()

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

Code/DriverTrace.py

C Averaging Results Driver

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 16 09:41:05 2019

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

doAveragePreemptive         = False
```

```
doAverageNonPreemptive      = False
doAverageNoPriority         = True

averagedOver = 150
linkCapacity = 1*10**6

if (doAverageNonPreemptive == True):
    print("#######################################################################")
    print("########              Averaged Non-preemptive queuing #######")
    print("#######################################################################")

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


    totalInterArrivalLow        = []
    totalInterArrivalHigh       = []
    totalTransmissionLow        = []
    totalTransmissionHigh       = []
    totalResponseLow            = []
    totalResponseHigh           = []
    totalAveragePacketSizeLow   = []
    totalAveragePacketSizeHigh  = []
    totalAverageDelaysLow       = []
    totalAverageDelaysHigh      = []
    totalArrivalRateLow         = []
    totalArrivalRateHigh        = []
    totalServiceRateLow         = []
    totalServiceRateHigh        = []
    allCombinedLengths          = [[]] * averagedOver
    allLengthsHighPriority      = [[]] * averagedOver
    allLengthsLowPriority       = [[]] * averagedOver

    for k in range(0, averagedOver):
        print(k)
        packets = [[],[]]
        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))

        # 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
        for i in range(0, len(interArrivalHigh)):
            highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH')
            packets[1].append(highPacket)

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

        if (k == 0):
            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 ----------------", len(packetSizeLow) + len(packetSizeHigh), "packets" )

        totalInterArrivalLow        += interArrivalLow
        totalInterArrivalHigh       += interArrivalHigh
        totalTransmissionLow        += transTimes[0]
        totalTransmissionHigh       += transTimes[1]
        totalResponseLow            += respTimes[0]
        totalResponseHigh           += respTimes[1]
        totalAveragePacketSizeLow   += packetSizeLow
        totalAveragePacketSizeHigh  += packetSizeHigh
        totalAverageDelaysLow       += queuingDelays[0]
        totalAverageDelaysHigh      += queuingDelays[1]

        startEnd = [[],[]]
        arrivalTimes = []
        startEndLow = [[],[]]
        arrivalTimesLow = []
        startEndHigh = [[],[]]
        arrivalTimesHigh = []

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

            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)

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

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

    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(
    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) , "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(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(totalAveragePacketSizeLow))/((sum(totalInterArrivalLow)))*10**6 + (len(totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh)))*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)*50000)/90000), 4), "packets/second")


    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]

    averagedCombinedLength = np.asarray(averagedCombinedLength)
    averagedLowPriorLength = np.asarray(averagedLowPriorLength)
    averagedHighPrioLength = np.asarray(averagedHighPrioLength)


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

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


    plt.figure()
    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 non-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 Mbps'))
    plt.legend(loc='best')
    plt.xlabel('Time (seconds)')
    plt.ylabel('Customers waiting in queue')
    plt.title('Averaged low and high non-preemptive queues')
    plt.show()


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

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


    totalInterArrivalLow       = []
    totalInterArrivalHigh      = []
    totalTransmissionLow       = []
    totalTransmissionHigh      = []
    totalResponseLow           = []
    totalResponseHigh          = []
    totalAveragePacketSizeLow  = []
    totalAveragePacketSizeHigh = []
    totalAverageDelaysLow      = []
    totalAverageDelaysHigh     = []
    totalArrivalRateLow        = []
    totalArrivalRateHigh       = []
    totalServiceRateLow        = []
    totalServiceRateHigh       = []
    allCombinedLengths         = [[]] * averagedOver
    allLengthsHighPriority     = [[]] * averagedOver
    allLengthsLowPriority       = [[]] * averagedOver

    for k in range(0, averagedOver):
        print(k)
        packets = [[],[]]
        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))

        # 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
        for i in range(0, len(interArrivalHigh)):
            highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH')
            packets[1].append(highPacket)

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

        if (k == 0):
            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 ----------------", len(packetSizeLow) + len(packetSizeHigh), "packets" )

        totalInterArrivalLow       += interArrivalLow
        totalInterArrivalHigh      += interArrivalHigh
        totalTransmissionLow       += transTimes[0]
        totalTransmissionHigh      += transTimes[1]
        totalResponseLow           += respTimes[0]
        totalResponseHigh          += respTimes[1]
        totalAveragePacketSizeLow  += packetSizeLow
        totalAveragePacketSizeHigh += packetSizeHigh
        totalAverageDelaysLow      += queuingDelays[0]
        totalAverageDelaysHigh     += queuingDelays[1]

        startEnd = [[],[]]
        arrivalTimes = []
        startEndLow = [[],[]]
        arrivalTimesLow = []
        startEndHigh = [[],[]]
        arrivalTimesHigh = []

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

            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)

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

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

    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(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) , "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(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(totalAveragePacketSizeLow))/((sum(totalInterArrivalLow)))*10**6 + (len(totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh)))*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)*50000)/90000), 4), "packets/second")
    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]

    averagedCombinedLength = np.asarray(averagedCombinedLength)
    averagedLowPriorLength = np.asarray(averagedLowPriorLength)
    averagedHighPrioLength = np.asarray(averagedHighPrioLength)


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

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


    plt.figure()
    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 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 Mbps'))
    plt.legend(loc='best')
    plt.xlabel('Time (seconds)')
    plt.ylabel('Customers waiting in queue')
    plt.title('Averaged low and high preemptive queues')
    plt.show()


if (doAverageNoPriority == True):
    print("##############################################################################")
    print("##########                Averaged No Priority queuing         #########")
    print("##############################################################################")
    time = np.arange(0, 101*10**6, 1000000)


    totalInterArrivalLow        = []
    totalInterArrivalHigh       = []
    totalTransmissionLow        = []
    totalTransmissionHigh       = []
    totalResponseLow            = []
    totalResponseHigh           = []
    totalAveragePacketSizeLow   = []
    totalAveragePacketSizeHigh  = []
    totalAverageDelaysLow       = []
    totalAverageDelaysHigh      = []
    totalArrivalRateLow         = []
    totalArrivalRateHigh        = []
    totalServiceRateLow         = []
    totalServiceRateHigh        = []
    allCombinedLengths          = [[]] * averagedOver
    allLengthsHighPriority      = [[]] * averagedOver
    allLengthsLowPriority       = [[]] * averagedOver

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

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

        # 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
        for i in range(0, len(interArrivalHigh)):
            highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH')
            packets[1].append(highPacket)

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

        if (k == 0):
            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 ----------------", len(packetSizeLow) + len(packetSizeHigh), "packets" )

        totalInterArrivalLow       += interArrivalLow
        totalInterArrivalHigh      += interArrivalHigh
        totalTransmissionLow       += transTimes[0]
        totalTransmissionHigh      += transTimes[1]
        totalResponseLow           += respTimes[0]
        totalResponseHigh          += respTimes[1]
        totalAveragePacketSizeLow  += packetSizeLow
        totalAveragePacketSizeHigh += packetSizeHigh
        totalAverageDelaysLow      += queuingDelays[0]
        totalAverageDelaysHigh     += queuingDelays[1]

        startEnd = [[],[]]
        arrivalTimes = []
        startEndLow = [[],[]]
        arrivalTimesLow = []
        startEndHigh = [[],[]]
        arrivalTimesHigh = []

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

            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)

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

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

    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(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) , "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(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(totalAveragePacketSizeLow))/((sum(totalInterArrivalLow)))*10**6 + (len(totalAveragePacketSizeHigh))/((sum(totalInterArrivalHigh)))*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)*50000)/90000), 4), "packets/second")
    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]

    averagedCombinedLength = np.asarray(averagedCombinedLength)
    averagedLowPriorLength = np.asarray(averagedLowPriorLength)
    averagedHighPrioLength = np.asarray(averagedHighPrioLength)


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

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


    plt.figure()
    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 -*-
"""
Created on Mon Sep 16 09:41:12 2019

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

doVaryingArrivalPreempt     = False
doVaryingArrivalNonPreempt  = True
doVaryingArrivalNoPriority  = False

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


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

    allQueuesLow  = []
    allQueuesHigh = []
    allQueuesComb = []

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

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


    totalInterArrivalLow       = []
    totalInterArrivalHigh      = []
    totalTransmissionLow       = []
    totalTransmissionHigh      = []
    totalResponseLow           = []
    totalResponseHigh          = []
    totalAveragePacketSizeLow  = []
    totalAveragePacketSizeHigh = []
    totalAverageDelaysLow      = []
    totalAverageDelaysHigh     = []
    totalArrivalRateLow        = []
    totalArrivalRateHigh       = []
    totalServiceRateLow        = []
    totalServiceRateHigh       = []
    allCombinedLengths         = [[]] * averagedOver
    allLengthsHighPriority     = [[]] * averagedOver
    allLengthsLowPriority      = [[]] * averagedOver

    for k in range(0, averagedOver):
        packets = [[],[]]
        transCap = linkCapacity
        print(l,k)
        # Read the csv file data into respective lists to store the data
        interArrivalHigh = list(np.random.exponential(arrRates[l] , int(100/(arrRates[l]*10**-6))))
        packetSizeHigh   = list(np.random.exponential(1000, int(100/(arrRates[l]*10**-6))))
        interArrivalLow  = list(np.random.exponential(2000 , 50000))
        packetSizeLow    = list(np.random.exponential(1000, 50000))

        # 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
        for i in range(0, len(interArrivalHigh)):
            highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH')
            packets[1].append(highPacket)

        # Get the parameters
        arrivalTimes,transTimes,queuingDelays,startEnd,respTimes, allPackets = calcNonPreemptive(transCap, packets)
        sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))), float(sum(packetSizeHigh))/len(packetSizeHigh)]
        if (k == 0):
            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 ----------------", len(packetSizeLow) + len(packetSizeHigh), "packets" )

        totalInterArrivalLow        += interArrivalLow
        totalInterArrivalHigh       += interArrivalHigh
        totalTransmissionLow        += transTimes[0]
        totalTransmissionHigh       += transTimes[1]
        totalResponseLow            += respTimes[0]
        totalResponseHigh           += respTimes[1]
        totalAveragePacketSizeLow   += packetSizeLow
        totalAveragePacketSizeHigh  += packetSizeHigh
        totalAverageDelaysLow       += queuingDelays[0]
        totalAverageDelaysHigh      += queuingDelays[1]

        startEnd = [[],[]]
        arrivalTimes = []
        startEndLow = [[],[]]
        arrivalTimesLow = []
        startEndHigh = [[],[]]
        arrivalTimesHigh = []

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

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

        allLengthsHighPriority[k] = deepcopy(highPriority)

        allCombinedLengths[k] = list( map(add, highPriority, lowPriority) )

    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(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) , "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(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(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[l]*10**-6)/(50000+arrRates[l]*10**-6)), 4), "packets/second")

     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]

     averagedCombinedLength = np.asarray(averagedCombinedLength)
     averagedLowPriorLength = np.asarray(averagedLowPriorLength)
     averagedHighPrioLength = np.asarray(averagedHighPrioLength)


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

     allQueuesComb.append(deepcopy(averagedCombinedLength))
     allQueuesHigh.append(deepcopy(averagedHighPrioLength))
     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(allQueuesHigh[a])/len(allQueuesHigh[a]), 4), "packets/second")

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

```

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

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

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

    allQueuesLow  = []
    allQueuesHigh = []
    allQueuesComb = []

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


        totalInterArrivalLow       = []
        totalInterArrivalHigh      = []
        totalTransmissionLow       = []
        totalTransmissionHigh      = []
        totalResponseLow           = []
        totalResponseHigh          = []
        totalAveragePacketSizeLow  = []
        totalAveragePacketSizeHigh = []
        totalAverageDelaysLow      = []
        totalAverageDelaysHigh     = []
        totalArrivalRateLow        = []
        totalArrivalRateHigh       = []
        totalServiceRateLow        = []
        totalServiceRateHigh       = []
        allCombinedLengths         = [[]] * averagedOver
        allLengthsHighPriority     = [[]] * averagedOver
        allLengthsLowPriority      = [[]] * averagedOver

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

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

```
        interArrivalHigh = list(np.random.exponential(arrRates[l] , int(100/(arrRates[l]*10**-6))))
        packetSizeHigh   = list(np.random.exponential(1000, int(100/(arrRates[l]*10**-6))))
        interArrivalLow  = list(np.random.exponential(2000 , 50000))
        packetSizeLow    = list(np.random.exponential(1000, 50000))

        # 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
        for i in range(0, len(interArrivalHigh)):
            highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH')
            packets[1].append(highPacket)

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

        if (k == 0):
            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 ----------------", len(packetSizeLow) + len(packetSizeHigh), "packets" )

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

        startEnd = [[],[]]
        arrivalTimes = []
        startEndLow = [[],[]]
        arrivalTimesLow = []
        startEndHigh = [[],[]]
        arrivalTimesHigh = []

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

            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)

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

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


    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(
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) , "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(
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(
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[l]*10**-6)/(50000+arrRates[l]*10**-6)), 4), "packets/second")

    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]
        averagedCombinedLength = np.asarray(averagedCombinedLength)
        averagedLowPriorLength = np.asarray(averagedLowPriorLength)
        averagedHighPrioLength = np.asarray(averagedHighPrioLength)


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

        allQueuesComb.append(deepcopy(averagedCombinedLength))
        allQueuesHigh.append(deepcopy(averagedHighPrioLength))
        allQueuesLow.append(deepcopy(averagedLowPriorLength))

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

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

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

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

    plt.figure()
    for t in range(0, len(allQueuesHigh)):
        plt.plot(time, allQueuesHigh[t], label="High " + str(arrRates[t]) + " us")
    plt.legend(loc='best')
    plt.xlabel('Time (seconds)')
    plt.ylabel('Customers waiting in queue')
    plt.title('Preemptive Varying arrRate High Priority')
    plt.show()
if( doVaryingArrivalNoPriority == True):
    print("################################################################")
    print("########            Averaged No Priority varying arrRate #######")
    print("################################################################")

    allQueuesLow  = []
    allQueuesHigh = []
    allQueuesComb = []

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


        totalInterArrivalLow       = []
        totalInterArrivalHigh      = []
        totalTransmissionLow       = []
        totalTransmissionHigh      = []
        totalResponseLow           = []
        totalResponseHigh          = []
        totalAveragePacketSizeLow  = []
        totalAveragePacketSizeHigh = []
        totalAverageDelaysLow      = []
        totalAverageDelaysHigh     = []
        totalArrivalRateLow        = []
        totalArrivalRateHigh       = []
        totalServiceRateLow        = []
        totalServiceRateHigh       = []
        allCombinedLengths         = [[]] * averagedOver
        allLengthsHighPriority     = [[]] * averagedOver
        allLengthsLowPriority      = [[]] * averagedOver

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

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

            # 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
            for i in range(0, len(interArrivalHigh)):
                highPacket = Packet(packetSizeHigh[i], interArrivalHigh[i], 'HIGH')
                packets[1].append(highPacket)

            # Get the parameters
            arrivalTimes,transTimes,queuingDelays,startEnd,respTimes,allPackets = calcNoPriority(transCap, packets)
            sizeAvg = [sum(packetSizeLow) / (float(len(packetSizeLow))), float(sum(packetSizeHigh))/len(packetSizeHigh)]
            if (k == 0):
                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 ----------------", len(packetSizeLow) + len(packetSizeHigh), "packets" )

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

            startEnd = [[],[]]
            arrivalTimes = []
            startEndLow = [[],[]]
            arrivalTimesLow = []
            startEndHigh = [[],[]]
            arrivalTimesHigh = []

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

                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)

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

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


        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(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) , "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(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(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[l]*10**-6)/(50000+arrRates[l]*10**-6)), 4), "packets/second")

        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]

        averagedCombinedLength = np.asarray(averagedCombinedLength)
        averagedLowPriorLength = np.asarray(averagedLowPriorLength)
        averagedHighPrioLength = np.asarray(averagedHighPrioLength)


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

        allQueuesComb.append(deepcopy(averagedCombinedLength))
        allQueuesHigh.append(deepcopy(averagedHighPrioLength))
        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(allQueuesHigh[a])/len(allQueuesHigh[a]), 4), "packets/second")

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

plt.legend(loc='best')
plt.xlabel('Time (seconds)')
plt.ylabel('Customers waiting in queue')
plt.title('No priority Varying arrRate Low LowPriority')
plt.show()

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

Code/DriverVary.py