CS4150: Computer Networks Lab

Lab6

111901030
Mayank Singla

Q1. Write a program called "rr" that implements round-robin queuing disciple. This function should take the service rate as the input argument. The file "arrivals.txt" contains the list of packet arrivals. Each packet arrival is denoted by a line in this file as "W X Y Z", where W is the packet arrival time, X is the packet ID, Y is the queue ID into which the packet arrives, and Z is the packet length. You should be able to invoke your program as follows:

```
./rr 4.0 < arrivals.txt
```

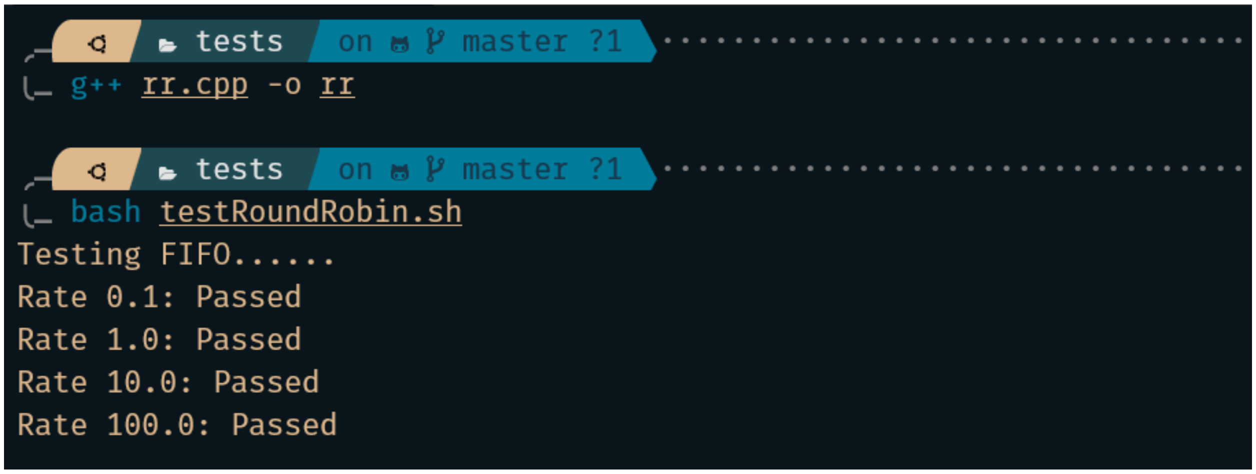
The output of the above invocation should be, for each packet, "U V" (without quotes), where U is the transmission time of the packet ("`%.2f`" format), and V is the packet ID.

I am creating a **Packet** class that represents a packet received as input having the fields **arrTime** for the arrival time of the packet, **packId** for the packet ID, **queueId** for the queue ID, and **packLen** for the length of the packet, and also a constructor for this class.
I am also creating a **Queue** class that represents a packet queue having the fields **queueId** for the queue ID and **packetQ** for representing the underlying packet queue. I am defining some constructors for this class and general purpose queue functionalities like **empty()**, **front()**, **push()** and **pop()**.

Next, I am defining the function **readInput** which reads the input packets and populates the packets into different queues corresponding to their queue Ids and also populates the variables start queue index and the number of packets taken as arguments by reference in this function. For keeping the queues in sorted order of their queue Ids, I am using a map to store the packets and then populating the vector of queues using this map.
Next, I am defining the function **getMinArrTimePackQueueIdx** which finds the index of the queue having the front packet with the minimum arrival time. It simply iterates over all the queues and finds the packet with the minimum arrival time. In case there are multiple packets with the same minimum arrival time, then the packet with the larger queue ID is given the preference. This observation was observed from the test cases to pass. This function will be used in the main algorithm for the round-robin queueing disciple.

Now, in my **main** function, I am first of all reading the **serviceRate** from the input and calling the **readInput** function to read the input and build all the queues with their input packets. Next, I am defining some variables to keep track of the current transmission time of the packet, which also serves as the transmission time for the last packet, a variable to keep track of the number of packets transmitted so far, a variable to keep track of the number of queues skipped in one round, a variable to keep track of the total number of queues in the input. Now, I am looping till the number of packets transmitted is not equal to the total number of input packets. Now, firstly, if it is the first packet, I am initializing the transmission time with the arrival time of the first packet. We will start with the queue for which the packet came first and whose index was given to us by the **readInput** function. Now, I am checking if the queue is empty and if it is, then I am skipping that queue and incrementing the number of queues skipped variable. Then, I am taking the current packet and if its arrival time is greater than the transmission time of the last packet, then I am again skipping this queue and incrementing the number of queues skipped variable. If the arrival time is less or equal to the transmission time of the last packet, then I am updating my transmission time by the time it takes to transmit the current packet which will be (packet length/service rate). If the packet is successfully transmitted then I am incrementing the number of packets transmitted and pop out the packet from the corresponding queue and reset the number of queues skipped variable to 0 and print the result. I am always incrementing the index of the current queue in a cyclic manner using the modulo operation with the total number of queues. If during the transmission, my number of queues skipped variable becomes equal to the total number of queues, that means I need to jump to the queue having the front packet with the minimum arrival time. I am calculating the index for this queue using the **getMinArrTimePackQueueIdx** function and resetting the transmission time with the arrival time for this packet and also resetting the number of queues skipped to 0.

Here are the results for executing the test cases given:



```
 tests  on  master ?1
╰─ g++ rr.cpp -o rr

 tests  on  master ?1
╰─ bash testRoundRobin.sh
Testing FIFO......
Rate 0.1: Passed
Rate 1.0: Passed
Rate 10.0: Passed
Rate 100.0: Passed
```

Q2. Write a program called "wfq" that implements weighted-fair queuing disciple. This function should take, as arguments, the service rate and the weights for each of the queues. The file "arrivals.txt" contains the list of packet arrivals. Each packet arrival is denoted by a line in this file as "W X Y Z", where W is the packet arrival time, X is the packet ID, Y is the queue ID into which the packet arrives, and Z is the packet length. You should be able to invoke your program as follows:

```
./wfq 4.0 1.0 1.0 1.0 1.0 < arrivals.txt
```

The output of the above invocation should be, for each packet, "U V W" (without quotes), where U is the transmission time of the packet ("`%.2f`" format), V is the packet ID, and W is the id of the queue that output this packet.

I am using the same **Packet** and **Queue** class as in the previous question just this time I have one more field in my **Queue** class which is the **weight** which indicates the weight given the queue in the weighted-fair queueing. Also, I am using the same **readInput** function as in the previous question and now this time, also populating the queues with their given weights received from the input arguments.
Next, in my **main** function, I am reading the **serviceRate** and the weights of the queues received as arguments and then populating all the queues by calling the **readInput** function. Then I am declaring a min priority queue of the pair **(finishTime, Packet)** using a custom comparator function of my own. The custom comparator function returns the packet with the lower finish time first and if for two packets the finish time is the same, then the packet with the lower packet ID is returned. Then, I am processing all the queues independently and keeping track of the finish time of a packet in the queue. Now, using the below formula, I am calculating the finish time for each packet in the queue and pushing the pair (finishTime, Packet) into the priority queue.

$$F_i = max(A_i, F_{i-1}) + \frac{L_i}{W}$$

where F_i = Finish time of the current packet

F_{i-1} = Finish time of the previous packet

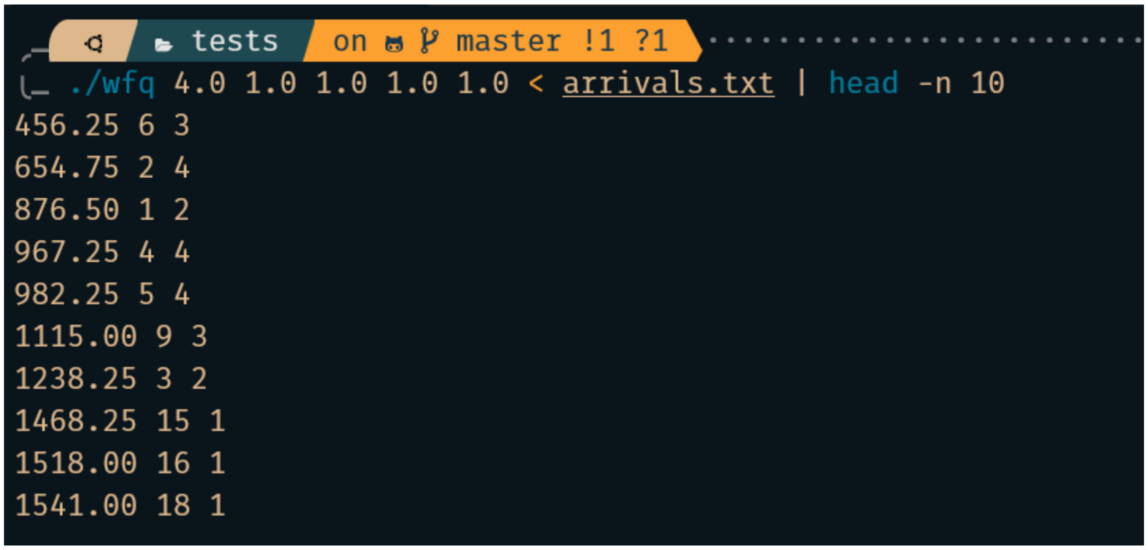
A_i = Arrival time of the current packet

L_i = Length of the current packet

W = Weight of the current queue

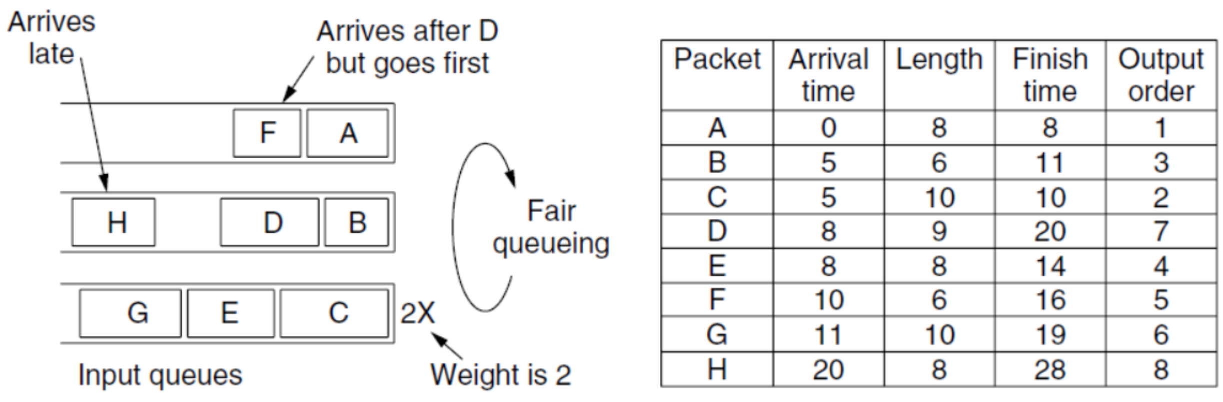
Then I am popping all the pairs from the priority queue and calculating the transmission time for each packet using the previous transmission time of the packet and the given service rate for the packet and printing the results in the desired format.

Here is the initial result of executing the command mentioned in the question.

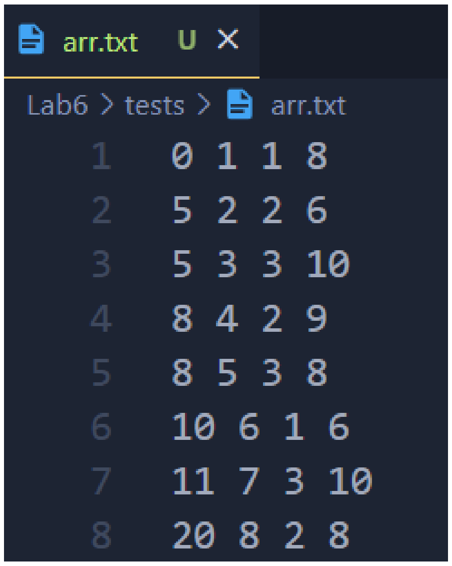


```
./wfq 4.0 1.0 1.0 1.0 1.0 < arrivals.txt | head -n 10
456.25 6 3
654.75 2 4
876.50 1 2
967.25 4 4
982.25 5 4
1115.00 9 3
1238.25 3 2
1468.25 15 1
1518.00 16 1
1541.00 18 1
```

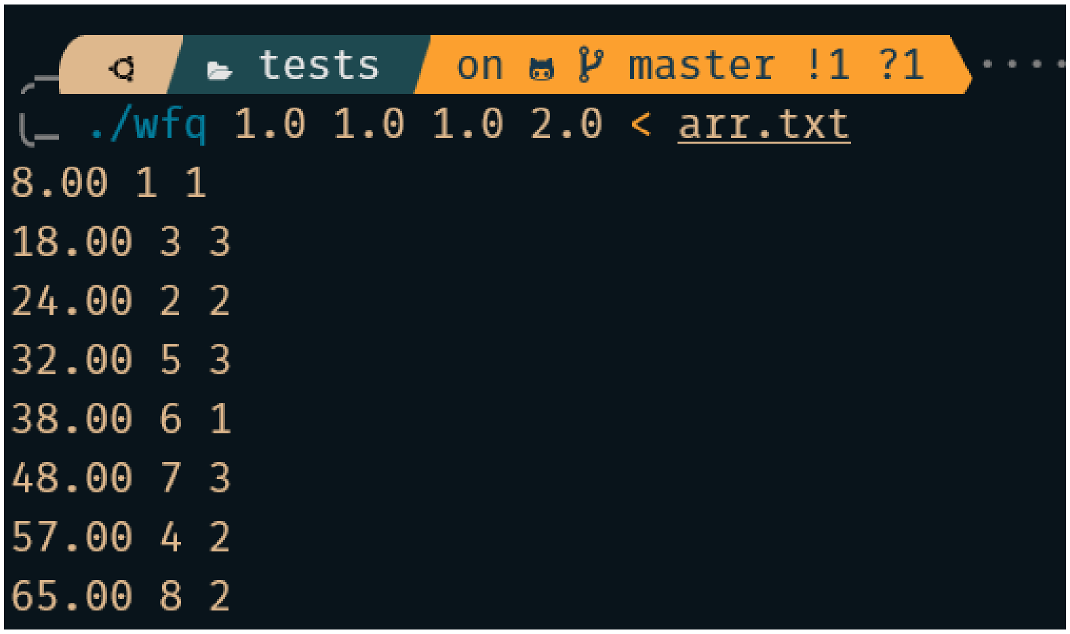
Here is the result of executing the above program for the example from the book.



Packet	Arrival time	Length	Finish time	Output order
A	0	8	8	1
B	5	6	11	3
C	5	10	10	2
D	8	9	20	7
E	8	8	14	4
F	10	6	16	5
G	11	10	19	6
H	20	8	28	8



```
0 1 1 8
5 2 2 6
5 3 3 10
8 4 2 9
8 5 3 8
10 6 1 6
11 7 3 10
20 8 2 8
```



```
tests on master !1 ?1
./wfq 1.0 1.0 1.0 2.0 < arr.txt
8.00 1 1
18.00 3 3
24.00 2 2
32.00 5 3
38.00 6 1
48.00 7 3
57.00 4 2
65.00 8 2
```