**Packet Scheduling**

**Total points: 10**

**Warning**: **Students are members of a learning community committed to the search for knowledge and truth. Essential to that search is the faithful adherence by all students to the highest standards of honesty and integrity. A grade of “0” or “F” will be assigned to assignments on which cheating, plagiarism or any other form of academic dishonesty is committed or determined to have occurred. Using source code for your assignments from anywhere is considered plagiarism/cheating and is therefore prohibited. For details, see Wright State University Student Handbook under “Academic Dishonesty”.**

**You must not attempt to get help from external sources such as online sites that offer solutions. Any submission for grading must be your own work. You must not copy from others or allow your work to be copied. We will check for similarities of code. Any violation will result in your receiving zero for this assignment. No exception.**

This project needs to be completed using Java and is an individual project. Naming of your programs must strictly follow the following convention: (1) one file for each part; (2) Lab2 followed by your Uid and I for part I (for example Lab2U00998499I.java). Upload only your source code (one file for each part) to the Pilot dropbox.

In class lectures, we discussed the packet scheduling algorithm – Generalized Processor Sharing (GPS). GPS cannot be directly applied in a real network for ordering packets for transmission because it assumes a fluid packet/traffic model. Real-world packets consist of bits/bytes, and they must be transmitted atomically, i.e., a packet must be transmitted as a single entity. Packet-by-packet GPS approximates the GPS scheduler closely using a virtual time-based implementation. Refer to **the paper** for details on GPS/PGPS: *Abhay K. Parekh and Robert Gallager, ``A generalized processor sharing approach to flow control in integrated services networks: the single node case,'' IEEE/ACM Trans. on Networking, Vol. 1, No. 3, pages 344--357, June 1993.*

A good object-oriented programming style for this lab is a consideration in grading. Therefore, you should design and implement your solutions carefully. Code must be commented. Your report must adequately describe your implementation. Test cases will exercise your programs thoroughly. Therefore, your programs should efficiently handle potentially large problems.

**Part I. Packet-by-packet GPS (PGPS)**

Your task in Part I is to implement the virtual-time based PGPS scheduling algorithm. In the lecture, we explained that the virtual time advance rate is variable, depending on the flows that are backlogged in virtual time.

In a PGPS scheduler, an arriving packet at time ***t*** (real time), arrives at a virtual time ***VT(t)*** where ***VT(t)*** is a virtual time function that maps real time ***t*** to its corresponding virtual time. The scheduler then calculates the virtual start time and virtual finish time of the packet using formula (11) on page 348 of the paper.

The packet is stamped with the virtual finish time (i.e., the packet carries the virtual finish time). Packets are ordered in increasing order of the virtual finish time in the packet queue, waiting to be transmitted. When the PGPS scheduler chooses the next packet to transmit at time ***t***, it selects among all the packets that are queued at ***t***, the first packet that has the minimum virtual finish time.

For the virtual time-based implementation to work, we need to keep track of the virtual time function ***VT(t)***. This is accomplished by noting that the set of backlogged flows does not change between two consecutive events. An event is a packet arrival, or a packet departure. There are two kinds of packet departure events: (1) packet departure in the PGPS scheduler (in real time) when a packet is transmitted; (2) packet departure in virtual time (we need to know when packets depart in virtual time to track the set of backlogged flows). A flow is considered backlogged if it has a packet to transmit in virtual time. The packet arrival event and the second type of departure event may change the set of backlogged flows and therefore impact the virtual time advancement rate.

Notice that there is no event (no arrivals and departures) between two consecutive events, and the set of backlogged flows will remain constant between 2 consecutive events. Let ***Bj*** be the set of backlogged flows during the (real time) interval (***tj-1*, *tj***). We will call the interval (***tj-1*, *tj***) the  ***jth****event interval.* There is a one-to-one correspondence between the real time clock value and the virtual clock time value, but the correspondence is rather complicated. The real time is denoted by ***t*** and the virtual time that corresponds to the real time ***t***is denoted by ***VT(t)****.* The time keeping is done as follows. The real time clock starts running when the first packet of a busy period arrives, i.e.: ***t1 = 0****.* This makes sense. Before the first packet arrival, the link is idle, and we do not to do any time keeping exercise. When the first packet of a busy period arrives, the virtual time clock also starts running: ***VT(0) = 0****.* So, the real time clock and the virtual time clock begin to run when the first packet of a busy period arrives. The virtual time changes with a constant rate within each event interval (***tj-1*, *tj***) in which ***Bj*** remains unchanged. The virtual time may advance with different rates in different event intervals. The virtual time advances in the   *jth event interval*using formula (10) on page 348 of the paper.

Also note one more time that the set ***B***of backlogged flows may change when a flow goes from idle to active when a packet of that flow arrives (in real time) or goes from being backlogged to idle in virtual time. However, we need to update the set ***B*** in real time. This is a challenge when a flow becomes idle because we need to map the virtual time (when the flow become idle) back to the real time at which time we can then update the set ***B***. Fortunately, we can map the virtual time when a packet departs to its corresponding real time given the packet’s virtual finish time. This is the time when ***B*** may change because a flow may become not backlogged when the last packet departs from the flow. Mapping from the virtual finish time of a packet to real time given a minimum virtual finish time ***Fmin*** of the packet that departs can be done using the formula that calculates ***Next(t)*** given on page 348 of the paper. At ***Next(t)*** the set ***B*** may be adjusted to remove the flow that is no longer backlogged.

**Suggestions of implementation**: as described above, we see that the PGPS scheduler must process events: packet arrivals, packet departures. This immediately suggests event-driven based implementation. You can define various classes, such as Packet, Flow, Event etc.

**INPUT FORMAT (file flows.txt):**

The first line of input contains an integer N (i.e., N flows of packets). The next N lines each contain the GPS weight for each flow. These are then followed by a line of input that contains an integer M (i.e., the number of packets that will arrive from the N flows). Each of the next M lines contains three numbers: time of packet arrival, flow id, and packet length. The packet length has been normalized by link transmission rate.

**OUTPUT FORMAT (file flowout.txt):**

Output the order of packet transmission using the PGPS scheduler. The specific format is shown in the sample output below.

**SAMPLE INPUT:**

6

0.5

0.1

0.1

0.1

0.1

0.1

13

0 1 1

0 2 1

0 3 1

0 4 1

0 5 1

0 6 1

1 1 1

2 1 1

3 1 1

4 1 1

5 1 1

6 1 1

7 1 1

**SAMPLE OUTPUT:**

nFlows = 6

nPackets = 13

Packet arrTime 0.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 0.0 flow id 2 w 0.1 packet length 1.0

Packet arrTime 0.0 flow id 3 w 0.1 packet length 1.0

Packet arrTime 0.0 flow id 4 w 0.1 packet length 1.0

Packet arrTime 0.0 flow id 5 w 0.1 packet length 1.0

Packet arrTime 0.0 flow id 6 w 0.1 packet length 1.0

Packet arrTime 1.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 2.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 3.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 4.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 5.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 6.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 7.0 flow id 1 w 0.5 packet length 1.0

Event{type=pgpsDeparture, t=1.0, p=Packet{flowId=1, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=2.0}}

Event{type=pgpsDeparture, t=2.0, p=Packet{flowId=1, arrTime=1.0, length=1.0, virtualStarTime=2.0, virtualFinishTime=4.0}}

Event{type=pgpsDeparture, t=3.0, p=Packet{flowId=1, arrTime=2.0, length=1.0, virtualStarTime=4.0, virtualFinishTime=6.0}}

Event{type=pgpsDeparture, t=4.0, p=Packet{flowId=1, arrTime=3.0, length=1.0, virtualStarTime=6.0, virtualFinishTime=8.0}}

Event{type=pgpsDeparture, t=5.0, p=Packet{flowId=4, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=6.0, p=Packet{flowId=1, arrTime=4.0, length=1.0, virtualStarTime=8.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=7.0, p=Packet{flowId=3, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=8.0, p=Packet{flowId=5, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=9.0, p=Packet{flowId=6, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=10.0, p=Packet{flowId=2, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=11.0, p=Packet{flowId=1, arrTime=5.0, length=1.0, virtualStarTime=10.0, virtualFinishTime=12.0}}

Event{type=pgpsDeparture, t=12.0, p=Packet{flowId=1, arrTime=6.0, length=1.0, virtualStarTime=12.0, virtualFinishTime=14.0}}

Event{type=pgpsDeparture, t=13.0, p=Packet{flowId=1, arrTime=7.0, length=1.0, virtualStarTime=14.0, virtualFinishTime=16.0}}

In the example above, a line that starts with “Event” gives the transmission of one packet at time *t*, *p* indicates the packet being transmitted.

**Part II. Worst-case Fair Fair Queuing (WF2Q)**

Part II of this lab builds on Part I. You should make sure that your Part I works correctly first.

In a WF2Q scheduler, when the link chooses the next packet at time ***t*** to transmit, it selects only from the packets queued that have started receiving service in the corresponding GPS scheduler at ***t****,* and furthermore chooses the packet among them that would complete service first in the corresponding GPS (i.e., with the smallest virtual finish time). Note that different from the PGPS scheduler, the WF2Q scheduler considers only those eligible packets and then chooses the packet with the minimum virtual finish time to transmit next.

Add/modify the code of Part I to implement the WF2Q scheduler. Submit the implementation as a separate Java file.

**INPUT FORMAT (file flows.txt):**

The format is the same as Part I.

**OUTPUT FORMAT (file flowout.txt):**

The format is the same as Part I.

**SAMPLE INPUT:**

The same sample input as Part I

**SAMPLE OUTPUT:**

nFlows = 6

nPackets = 13

Packet arrTime 0.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 0.0 flow id 2 w 0.1 packet length 1.0

Packet arrTime 0.0 flow id 3 w 0.1 packet length 1.0

Packet arrTime 0.0 flow id 4 w 0.1 packet length 1.0

Packet arrTime 0.0 flow id 5 w 0.1 packet length 1.0

Packet arrTime 0.0 flow id 6 w 0.1 packet length 1.0

Packet arrTime 1.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 2.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 3.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 4.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 5.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 6.0 flow id 1 w 0.5 packet length 1.0

Packet arrTime 7.0 flow id 1 w 0.5 packet length 1.0

Event{type=pgpsDeparture, t=1.0, p=Packet{flowId=1, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=2.0}}

Event{type=pgpsDeparture, t=2.0, p=Packet{flowId=3, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=3.0, p=Packet{flowId=1, arrTime=1.0, length=1.0, virtualStarTime=2.0, virtualFinishTime=4.0}}

Event{type=pgpsDeparture, t=4.0, p=Packet{flowId=4, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=5.0, p=Packet{flowId=1, arrTime=2.0, length=1.0, virtualStarTime=4.0, virtualFinishTime=6.0}}

Event{type=pgpsDeparture, t=6.0, p=Packet{flowId=2, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=7.0, p=Packet{flowId=1, arrTime=3.0, length=1.0, virtualStarTime=6.0, virtualFinishTime=8.0}}

Event{type=pgpsDeparture, t=8.0, p=Packet{flowId=5, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=9.0, p=Packet{flowId=6, arrTime=0.0, length=1.0, virtualStarTime=0.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=10.0, p=Packet{flowId=1, arrTime=4.0, length=1.0, virtualStarTime=8.0, virtualFinishTime=10.0}}

Event{type=pgpsDeparture, t=11.0, p=Packet{flowId=1, arrTime=5.0, length=1.0, virtualStarTime=10.0, virtualFinishTime=12.0}}

Event{type=pgpsDeparture, t=12.0, p=Packet{flowId=1, arrTime=6.0, length=1.0, virtualStarTime=12.0, virtualFinishTime=14.0}}

Event{type=pgpsDeparture, t=13.0, p=Packet{flowId=1, arrTime=7.0, length=1.0, virtualStarTime=14.0, virtualFinishTime=16.0}}