Computer Science 1 - Program 4
Drop Tail Routers
Assigned: Friday 3/23/2012
Due: Friday 4/6/2012 BY 11:55pm (Submit on Webcourses)

Objectives:
Use Queues and understand their different implementations, an array based implementation and a linked list based implementation.

Problem Description:
In times of high volume, a router may receive more data than it can process. Routers use a queue management algorithm to make sure that data is not lost. The simplest of these algorithms is known as Drop Tail. In Drop Tail, if the queue is filled to its maximum capacity, the newly arriving packets are dropped until the queue has enough room to accept incoming packets.

When the sender node discovers that a packet has not been successfully sent, it will resend the missing packet during the next clock cycle. In a heavily congested network this could occur multiple times until the missing packet has been successfully acknowledged from the router.

For this assignment you are tasked with creating a simulator for a network that uses a Drop Tail router.

There are a number of nodes (ip addresses) on your network that connect to the router. Nodes send requests directly to one another to signal a need to transmit data over the network. When data is transmitted, it passes from the sending node to the router and then to the destination node.

Each node may receive multiple requests for data. The node must processes the first request before it may process any additional requests. Therefore, the node uses a queue to manage its requests. For this assignment there is no limit on the amount of requests a node may receive.

Typically when data is sent it is broken up into many packets. In this assignment, the sending nodes evenly divide the data being sent into 1MB packets, and each request is guaranteed to be a whole number of megabytes. Further more, the router is only capable of queuing 20 of these packets at a time in its internal buffer.

1. The router must be implemented using an array based queue (PacketQueue below).
2. The nodes must be implemented using a linked list based queue (RequestQueue below).
3. Both queues must implement the following functions:
    a. Init (initialize queue)
    b. Enqueue
    c. Dequeue

Implementation:

You must use the exact structs as follows:

```
typedef struct Packet {
      char ipSender[20];
      char ipDest[20];
      int id;                 // The id is the current out of total
} Packet;

struct PacketQueue {
    Packet** elements;
    int front;
    int numElements;
    int queueSize;
};

typedef struct Request {
      char ipSender[20];
      int totalPackets;
} Request;

struct node {
    Request* request;
    struct node* next;
};

struct RequestQueue {
    struct node* front;
    struct node* back;
     int numElements;
};

// Each network device has a Request queue
typedef struct NetworkDevice {
      char ip[20];
      struct RequestQueue* RequestQ;      // Requested by this device
} NetworkDevice;
```

Input File Specification (Simulator.in):

The input file has a single positive integer, ***d***, on its first line, specifying the number of total days to run the simulation.

Each simulation is specified as follows:

- The first line of each simulation will contain a single positive integer, ***r***, representing the number of requests for that day of the simulation.
- The next line will store the number of nodes, ***n***.
- The following ***r*** lines will contain information about each request, listed in the order each request is received.
- The first piece of information on each of these lines will be the ipAddress of the requester (destination), followed by the ipAddress of the node it is requesting data from (sender).
- The final piece of information on each line is the number of MB the destination node is requesting from the sender (a positive integer less than 10,000)

You cannot begin transmitting data from a node to the router until all of the requests are read into their respective node queues.

Once the data is read in, you may begin a simulation. Each simulation will last for an unknown number of clock cycles (ms). The simulation ends once the final request is delivered.

During each clock cycle you may do the following:

- Transmit a single packet (1MB) from ***each*** sending node to the router
- The router may transmit a single packet (1MB) to the destination node

Once the router's queue is full (capacity 20 packets), it cannot accept additional packets from the sending nodes.

Sample Input File(Simulator.in):

```
2
4
4
192.168.1.2 192.168.1.3 5
192.168.1.3 192.168.1.4 10
192.168.1.4 192.168.1.5 10
192.168.1.5 192.168.1.2 5
8
4
192.168.1.2 192.168.1.3 5
192.168.1.3 192.168.1.4 10
192.168.1.4 192.168.1.5 10
192.168.1.5 192.168.1.2 5
192.168.1.2 192.168.1.3 5
192.168.1.3 192.168.1.4 10
192.168.1.4 192.168.1.5 10
192.168.1.5 192.168.1.2 5
```

Sample Output File(Simulator.out):

```
Day #1:

Device at 192.168.1.2 has received its request at time = 17ms
Device at 192.168.1.5 has received its request at time = 20ms
Device at 192.168.1.3 has received its request at time = 29ms
Device at 192.168.1.4 has received its request at time = 30ms

Day #2:

Device at 192.168.1.2 has received its request at time = 17ms
Device at 192.168.1.5 has received its request at time = 20ms
Device at 192.168.1.2 has received its request at time = 29ms
Device at 192.168.1.3 has received its request at time = 32ms
Device at 192.168.1.3 has received its request at time = 42ms
Device at 192.168.1.4 has received its request at time = 46ms
Device at 192.168.1.4 has received its request at time = 56ms
Device at 192.168.1.5 has received its request at time = 60ms
```

<u>Output Specification(Simulator.out):</u>
For each input case, the output should be of the following format, printed to Simulator.out:

Day #d:

```
Device at X has received its request at time = Yms
Device at X has received its request at time = Yms
```

Day #d+1:

etc.

where d is the day of the simulation, starting with day 1. X is the ipAddress of the receiving device, and Y is the time X received a packet that completed a request.

<u>Deliverables</u>
<u>Simulator.c, PacketQueue.h, and RequestQueue.h</u> turned in through WebCourses.
Make sure your Simulator.c #include's both PacketQueue.h and RequestQueue.h.