```
/*
   Computer Networks Laboratory (Lab) 15CSL77
   10. Write a program for congestion control using Leaky bucket algorithm.
*/

/* Network layer

   Varialbe traffic rate

   Match rate of output link/line

   Remove burstiness or jitter
*/

/* MTU (Maximum Transmission Unit) for Ethernet is 1500 bytes.
*/

/* Multiple approaches -

   Assume packet size is fixed
   Fixed packet size, variable number or packets,
     hence bucket capacity expressed in number of packets
   or
   Variable packet size, hence bucket capacity in terms of size of packets
*/

/* Compare to Dam, compare water to packets
     Water from catchment area - Inflow of packets into bucket
     Assume if overflow water is not sent downstream
       Then use / open spillway, main Gate - for overflow
               drop packets
     And constant outgoing flow rate - Bottom outlet
               Bucket, output, leak rate
*/

/* The leaky bucket consists of a finite queue

   When a packet arrives,
     if there is room on the queue
         Appended to the queue
     else
         Discard the packet as no space in bucket

   Assume leak rate is one packe every second, then
   At every clock tick one packet is transmitted
       Unless the queue is empty, called
*/

/*
   Router A to B
   A has queue so link to B does not get congested
   A is restricting its output

   A sends, fails, retries, instead match the capacity of A to B link

   A limits to match B
*/

/* Queue, enqueue, dequeue, isFull
   Circular queue
*/

/* Read parameters of the bucket:

   1. Read Bucket(Queue) Size, bucketSize or queueSize
      Size in terms of packets it can hold

   2. Read output rate, bucketRate
      In terms of packets Sent/output(dequeued) from bucket(queue)
```

```c
      Read number of times to simulate, time

      for t  ← 0 to time

         Simulate flow into bucket, incoming traffic

         Read number of incoming packets, numberOfIncomingPackets

         Read content of incoming packets, contentOfIncomingPacket[]

         Now add to bucket

         for packet p in numberOfIncomingPackets

            if bucket(queue) not full
               enqueue( contentOfIncomingPacket[p] )

            else
               packet p discarded

         Adding to bucket complete, now simulate flow out of bucket

         for  t  ← 0 to bucketRate                      Leaky bucket
            outgoingPacket = dequeue( contentOfIncomingPacket )
            print outgoingPacket transmitted
*/

/* Queue
   Queue size/capacity, front, rear, present/current size/capacity
*/

#include <stdio.h>    // Bucket == Queue

int main()              // Assume packets have fixed size, and
 {                      //   Bucket size == number packets bucket can hold
   int bucketSize;              // Maximum number of packets the bucket can hold
   int totalPacketsInBucket=0; // Keeps count of total number of packets in bucket
   int bucket[100];   // Array for bucket/queue, assume it would be big enough
   int front = 0;              // Front of bucket/queue
   int rear = -1;              // Rear of bucket/queue
   int leakRate;               // Packets let out of bucket per second
   int time;                   // Unit time instances the network traffic is simulated
   int numberOfInComingPacketsAtTimeT;
   int i;
   int j;
   int k;

   printf("\n Assume maximum packets the bucket can hold is bucket size.");
   printf("\n Enter the bucket size : ");    scanf("%d", &bucketSize);

   printf("\n Assume packets the bucket leaks out every time unit is leak rate");
   printf("\n Enter the leak rate : ");    scanf("%d", &leakRate);

   printf("\n Enter number of time unit instances to simulate : ");
   scanf("%d", &time);

   printf("\n Leaky bucket: ");
   for( i=0; i<time; i++ )
    {
      printf("\n\n Time t = %d\n   Enter the number of incoming packets: ", i);
      scanf("%d", &numberOfInComingPacketsAtTimeT);

      // If OpenMP (Open Multi-Processing) omp.h is used, and threading is enabled
      //      then, both enqueuing and dequeuing can be done simultaneously
      //      which resembles real life inflow and outflow of packets from bucket

      printf("\n     Enter the content of incoming packets at time t = %d\n", i);
```

```c
        for( j=0; j<numberOfInComingPacketsAtTimeT; j++ )         // Add to bucket
         {
          if( totalPacketsInBucket < bucketSize ) // If space in bucket, then
            { // Read the content and enqueue in bucket, content is just to
              printf("       Enter content of packet: ");   // distinguish packets
              scanf("%d", &bucket[++rear]);  // Add packet to rear of queue
              totalPacketsInBucket++;        // Increment number of packets in bucket
            }
          else
            { // totalPacketsInBucket == bucketSize, cannot add packet into bucket
              printf("\n       Bucket Overflow, drop the packet");
            }
         }

        printf("\n\n     Outgoing packets at time t = %d are ", i);
        for( j=0; j<leakRate; j++ ) // Remove leakRate number of packets from bucket
         {
          if( totalPacketsInBucket == 0 )
            {
              printf("\n       Bucket empty, underflow, no packets to leak out");
            }
          else
            { // Remove from front of bucket/queue; Increment front
              printf("\n       Packet with content %d leaked out", bucket[front++]);
              totalPacketsInBucket--;    // and decrement number of packets in bucket
            }
         }
      }

    return 0;
 }

// improve implementation with Circular Queue
/* circular queue
    front = -1
    rear = -1

    full if ( front == 0 && rear == size-1 ) or ( rear == (front-1)%(size-1) )

    empty if ( front == -1 ) or ( rear+1 % size == front )

    enqueue
      if front == -1 , front = 0
      queue[ rear+1 % size ] = value

    dequeue
      queue[front]
      front = front+1 % size

    display queue
*/

// improve - randomize bucket size, time, number and content of packets using
/* rand, rand_r, srand - pseudo-random number generator

      #include <stdlib.h>

      int rand(void);
      or
      int rand_r(unsigned int *seedp);
*/

/* Textbook:
    Behrouz Forouzon - Data Communications and Networking, McGraw Hill Edition
    Andrew S. Tanenbaum - Computer Networks
*/

/* Output:
```

```
Assume maximum packets the bucket can hold is bucket size.
Enter the bucket size : 5

Assume packets the bucket leaks out every time unit is leak rate
Enter the leak rate : 3

Enter number of time unit instances to simulate : 2

Leaky bucket:

Time t = 0
  Enter the number of incoming packets: 1

    Enter the content of incoming packets at time t = 0
      Enter content of packet: 101


    Outgoing packets at time t = 0:
      Packet with content 101 leaked out
      Bucket empty, underflow, no packets to leak out
      Bucket empty, underflow, no packets to leak out

Time t = 1
  Enter the number of incoming packets: 7

    Enter the content of incoming packets at time t = 1
      Enter content of packet: 103
      Enter content of packet: 104
      Enter content of packet: 106
      Enter content of packet: 107
      Enter content of packet: 109

      Bucket Overflow, drop the packet
      Bucket Overflow, drop the packet

    Outgoing packets at time t = 1:
      Packet with content 103 leaked out
      Packet with content 104 leaked out
      Packet with content 106 leaked out
*/
```