Programming Assignment #4
Ordered Delivery of Packets

May 3, 2017

1 Objective

- To implement a simple version of ordered delivery service required in many networking protocols and applications.

2 Submission Instruction

You are expected to submit using the online submission system using the upload file(s) link.

The submitted code file should be named a4.zip including

- **a queue implementaiton (queue.h)**
- **your driver a4.cpp**

if your file has a different name, it will not be considered in the evaluation.
The files should be directly included in the zip file not in a folder inside the zip file.

Submission Deadline is May 21 @ Midnight.

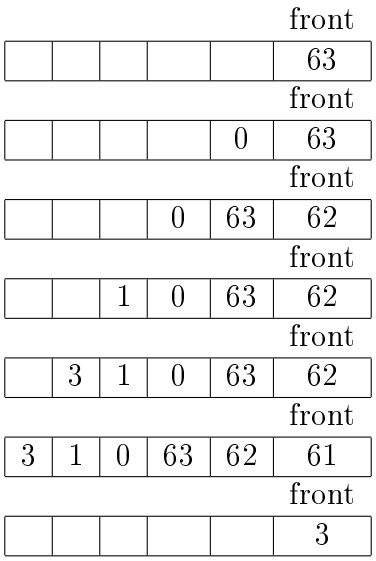
- Missing the deadline == No Marks for this assignment
- Submit even if your code is partially working
- Write the code yourself. Plagiarism (code copying) in any of the assignments ==> **-10 Marks**

3 Assignment Overview

- In this assignment, we will use a special type of queues which have a simple dequeue and an O(N) enqueue that decide where to insert the received object (similar to Priority Queues).
- This assignment is an application of a well-known issue in communications networks.
- In communication networks, a sender can send a sequence of packets (i.e. units of information transported over the network) but due to unpredictable network conditions, these packets can be lost or arrive out-of-order at the receiver.
- We focus here on the case of out-of-order delivery of packets at the receiver. The receiver has a buffer in which it temporarily stores the incoming packets. It only delivers packets that are in proper sequence to the final application that needs the information sent by the sender. For example assume the sender sends packets 0, 1, 2, ..., 9. The receiver receives the packets in the following time instants:

Time	Recieved Packet	Comments
1	2	
2	0	
3	1	0, 1, 2 are now in order -> Deliver packet 0, 1, 2
4	3	3 in order with 0,1,2 -> Deliver 3
5	5	
6	5	(duplicate reception, drop)
7	4	4 and 5 are inoder -> Deliver 4, 5
8	4	(duplicate reception, drop)
9	7	
10	8	
11	7	(duplicate reception, drop)
12	6	6, 7, 8 are inorder -> Deliver 6, 7, 8

- The receiver is initialized with the sequence number of the first packet expected to be recieved (initSeq) and a maximum number of outstanding packets it can store (this is known as window size in networks terminology).
- The sequence number is represented by a finite number of bits (seqNumBits). So for example, if the number of bits of the sequence number is 6, then we can have the sequence space 0, 1, . . . , 63. In this case, we define a parameter called MAXSEQNU which will be equal to 64.
- The sequence number can wrap and restart from 0. So, the sequence of 61, 62, 63, 0, 1 is actually perfectly in sequence.
- The window size is equal to 2^(winSizeBits) and has a maximum of 2^(seqNumBits-1).
- The received packets will be en-queued in the queue in a correct order, and once a sequence of expected ordered packets are received, they are removed from the queue to be delivered to the application. For example, if seqNumBits equals 6 and initSeq equals 61, and the received packets are 63, 0, 62, 1, 3, 61 respectively, the queue must store these packets in that order



N.B. At the last step, a sequence of expected ordered packets are received 61:1, so they are dequeued.

- When a set of inorder packets are delivered to the application, the next expected received packet is set to the sequence number of the last delivered inorder packet + 1 (Modulu MAXSEQNU). (In the previous example, the next expected packet is 2)
- If the receiver receives a packet with sequence number outside of allowable sequence space, then it is dropped. For example, if the receiver's window size is 32 (winSizeBits = 5) and the next expected received packet is 2, so the allowable sequence space is 2:33, if packet 34 is received, it will be dropped. The window is slided after receiving a correctly ordered new packet. If packet 2 is received, then the new window is 3:34. However, if packet 4 is received instead of 2, the window is not slided until the arrival of 2. This video visualizes the window concept.

4 Typical Operation

4.1 Input and Output lines

- In all the following, a4.exe is assumed to be the name of your executable file
- Input line include
  - a4.exe seqNumBits winSizeBits initSeq : pid1 pid2 .....
    - where seqNumBits represents the number of bits used by the sequence number
    - 2^(winSizeBits) represents the window size.
    - initSeq represents the first packet expected to be received
    - pidx are packet ids received by the end host.
  - OUTPUT: should be R [list of ordered received packet IDs] E [next expected packet ID] W [ordered list of out-of-order packets] D# [number of dropped packets]

4.2 Error Handling

- You should check for the correctness of every command
  - Wrong seqNumBits, winSizeBits or initSeq should print "**Invalid Configuration**"
  - Wrong packet ID should print "**Invalid Packet**"

4.3 Example Test Cases

- Input: a4.exe 4 3 3 : 3 5 4 9 7 7 4 6 10 15 0
  Output: R 3 4 5 6 7 E 8 W 9 10 15 D# 3
  - R 3 4 5 6 7 indicates the reception of packets 3 to 7
  - E 8 indicates that the application is expecting packet 8
  - W 9 10 15 indicates that 9, 10 and 15 are out of order packets waiting for delivery
  - D# 3 indicates that 3 packets are dropped: repeated packets 7 and 4, and out-of-window packet 0.
- Input: a4.exe 4 3 3 : 3 4 5 4 4 9 6 7 8
  Output: R 3 4 5 6 7 8 9 E 10 W D# 2
- Input: a4.exe 6 4 58 : 59 58 60 61 63 0 2 4 32
  Output: R 58 59 60 61 E 62 W 63 0 2 4 D# 1
- Input: a4.exe 4 3 3 : 4 5 6 7 a 4
  Output: Invalid Packet
- Input: a4.exe 4 3 3 : 4 5 6 7 100 4
  Output: Invalid Packet
- Input: a4.exe 4 3 100 : 4 5 6 7 a 4
  Output: Invalid Configuration (Higher priority)