CS6040 – Router Architectures and Algorithms, Jul.-Nov. 2024
Lab 4 – Packet Scheduling Algorithms
Due Date: Oct. 25, 2024, 11 PM, BY Moodle
Late submission Policy: 8% per 12-hour duration for a max. of 48 hours

1 Introduction

The objective of this assignment is to implement the Weighted Fair Queuing (WFQ) as described in Class Notes/Keshav's book, and study its performance.

The system consists of a set of N sources, where each source i generates traffic defined by the parameters: P^i, average number of packets generated per unit time, source's weight, and $\{L^i_{min}, L^i_{max}\}$ denote the minimum and maximum packet lengths (in packet length units) generated by source i. The length of a packet is **uniformly** distributed between $\{L^i_{min}, L^i_{max}\}$. The packet arrival process is exponential, i.e. the inter-arrival time between two packets from a source is exponentially distributed. The generated packets are sent to a single queue, that is serviced by the scheduling algorithm.

Consider a packet that is 1,000 packet units long; and let the output link process packets at the rate of 100,000 packet length units per unit time. Then, the transmission time for this packet is given by $\frac{1,000}{100,000} = 0.01$ time units.

The program will be invoked as follows:

```
% ./wfq -in inputfile -out outputfile
```

2 Inputs

The input parameters are: N, T: the total simulation time, C: the output link's packet processing capacity (in packet length units per unit time), B: Queue capacity (packets of maximum possible length) – this is the maximum number of packets that can be queued, from ALL the connections. This is followed by the per-source parameters, including packet length specifications. A single input file will contain these values in the following format:

```
N=4 T=1000000 C=100000 B=100
10 1000 1500 4 0.03 0.8
20 500 1200 3 0.02 0.7
20 750 1500 2.5 0.05 0.9
100 1000 1800 1.5 0.01 0.9
```

Each line, starting from the second line, specifies a source's parameters: $P^i, L^i_{min}, L^i_{max}, w_i, t^b_i, t^e_i$. Thus, in the above example, Line 1 specifies N, T, etc.; Line 2 specifies Source 1's parameters (10 packets per unit time, packet length uniformly distributed between 1000 and 1500 packet length units); absolute weight of this Source; Source i begins generating packets at $t^b_i * T$ units and stops generating packets at $t^e_i * T$ units. In the above example, source 4 starts generating at 10,000 time units and stops generating packets at 900,000 time units.

The effective relative weight of a source is given by:

$$ew_i = \frac{w_i}{\sum_{j=1}^{n} w_j}$$

When a packet from connection i arrives to a buffer that is full, the system drops the packet from connection i with the *smallest finish number*, in case of WFQ.

3 Output

The outputs that are expected from the program are:

1. System-Level Performance Metrics: (i) Server Utilization (averaged over the simulation duration), (ii) Fairness Index (Based on Jain's fairness index, `https://en.wikipedia.org/wiki/Fairness_measure`) and Related Fairness Bound (RFB) computed every 10000 time units, (iii) Average packet delay (across all packets) and (iv) Packet Drop Probability (across all connections).

2. Per-Source metrics:

Conn. ID	Total Packet Length Units Generated (B_G)	Total Packet Length Units Transmitted (B_T)	$\frac{B_G}{B_T}$	Fraction of link bandwidth allotted	Avg. Pkt Delay	Packet drop prob.
1						
2						
3						
...						
N						

4 What to Submit

A technical report that analyses the performance of the algorithm is required. You should generate at least two different input files, one for $N = 8$ and $N = 16$. In the report, you must discuss how the algorithm performs in terms of fairness, average delay, etc.

Please submit a tar-gzipped file including the source code files, sample output files, technical report, README, Makefile and COMMENTS files.

No README: -5%; No Makefile: -5%; No Sample Output Files: -10%

5 Grading

Traffic Gen. & Common Queuing:	10%
Scheduling:	70 %
Report:	15 %
Viva:	5 %

6 Implementation Notes

- Threads based: Each source can be implemented using a thread. The thread will be in a simple loop: (a) generate packet and add to the common queue, (b) go to sleep based on inter-packet arrival; (c) wake up and repeat from Step (a).

  The scheduler, which will compute the finish numbers, will be implemented as a separate thread. The scheduler will be responsible for selecting the next packet in the queue for scheduling and subsequently deleting. The common queue to which packets are added and deleted from, should be protected using a lock/mutex and/or counting semaphores. Note that this essentially is the multiple producer-single consumer problem.

  Implement the solution in phases: first, implement the packet generation phase, then the queuing phase (taking into account mutual exclusion), etc.

- In order to generate the inter-arrival time using the exponential distribution with parameter μ, sample C++ code is given below:

```
#include <iomanip>
#include <random>
#include <iostream>

// Compile: g++ -o rexp rexpgen.cpp
// Run: ./rexp <mu> <N>

using namespace std;

// https://cplusplus.com/reference/random/exponential_distribution/
int main (int argc, char *argv[])
{
  double exp_dist_mu = 100.0; // Mean inter-arrival is 1 / mu.
  int N = 100;

  if (argc > 1){
    exp_dist_mu = stod(argv[1]);
    N = stoi(argv[2]);                // Number of experiments
  }

  std::default_random_engine gen;
  std::exponential_distribution<double> MY_EXP_RNG (exp_dist_mu);

  double total = 0;
  // Generate  inter-packet arrival times
  for (int i = 0; i < N; ++i)
    {
      double sample = MY_EXP_RNG (gen);
      total += sample;
      cout << i << " " << sample << endl;
    }

  cout << "Experiment Mean is: " << total / N <<
    "\tTheoretical Mean is: "  << 1 / exp_dist_mu <<
    endl;
}
```

- The same can be attempted using a discrete event simulator such as:

  SimJava[1], SimPy[2], ns3[3], OMNET++[4], etc.

- ChatGPT or other AI tools may not be used for code generation. If such tools are used, it must be explicitly declared in the README file.

[1]http://www.dcs.ed.ac.uk/home/hase/simjava/
[2]https://simpy.readthedocs.io/en/latest/
[3]https://nsnam.org/
[4]http://www.omnetpp.org/

7 Optional (No Extra Credit)

Implement WRR, SCFQ, and DRR approaches for the same inputs and compare the fairness achieved by the different schemes.