### Modeling Discrete-Event Systems Using DEVS

### (2016 Fall)

### Assignment1: Non-Preemptive Priority Queuing Packet Scheduling Simulator

**NAGA PRASANTHI BOBBILI**

**STUDENT ID 101057215**

**CARLETON UNIVERSITY**

**Systems and Computer Engineering**

**PART I**

**Non-Preemptive Priority Queuing Packet Scheduling**

Packet scheduling refers to the decision process used to choose which packets should be serviced or dropped in a network while routing them. There are various techniques in routing the packets. Packet scheduling includes FIFO, Priority queuing, Fair queue, Waited queue etc. This model simulates Priority queuing system. Though it contains High, Medium, Normal and Low priority levels, this model talks only about High and Low to keep the simulation simple. This can be extended in future for other levels of priority.

The priority queuing, services the incoming data packets based on their priority. The Non preemptive priority queuing doesn’t interrupt the service if it has started though the high priority data packet request had come from the classifier. The classifier classifies the packet if it is of high priority or low priority based on the interface to which it is destined and the protocol types. Once the classification is done, the classifier sends the priority packets to the respective queues and even checks if the queue is full or empty. If it is full, then packet will be dropped at the tail of the queue and a new request for another packet will be raised. If the queue is empty then it will be sent to the scheduler which again checks for the transmit queue status and sends the packet. This is how the Priority Queuing System works.

The simulator model here contains total 5 components, 3 are atomic models and a coupled model with 4 models within the PQ system.

The *packet\_in\_n* is the number of packets to be sent through the network. The sequence of packets to the classifier will be sent and the classifier moves from *passive* state to *active* state once the packet is received. The classifier receives the packet and classifies as high or low by assigning it to the respective output port. This takes some time “*t\_c*”. If the packet is of high priority, it will be sent to “*High Priority Queue*” else to “*Low Priority Queue*”. Both will be in passive state until the packet arrives. “*High Priority Queue”* or “*Low Priority Queue*” will be checked for full or empty. If empty, the packet will be in queue and then “*pkt\_high\_rec*” will be sent to “*ReceiverH*” after some time “*t\_h*”. Same is the case with “*Low Priority Queue*”. If the queue is empty, the packet “*pkt\_low\_o*” will be in queue for some time and then sent to the scheduler after “*t\_l*” time. The “*ReceiverL*” transits from passive to active state and then outputs the packet after some time “*t\_s*”. Until the number of packets are transmitted through the queue, simulation will have to run.

# Part II

As shown in Figure 1, the PQ Simulator has 2 input and 2 outputs. The *packet\_in\_n*  input indicates the number of packets should be sent. The 2 outputs show the time when a packet has been sent out successfully (*packetSent*) from both the queues. The PQ Simulator consists of 4 components: Classifier, PQ System and two receivers. The classifier sends packets to the receiver and receives acknowledgement from the receiver. The PQ System contains two components, High Priority Queue and a Low Priority Queue.

**Formal Specifications**

The formal specifications <S, X, Y, δint, δext, λ, ta> for the atomic models are defined as follows:

**ReceiverH:**

S = {passive, active}

X = {in}

out

in

in

out

Y = {out}

δint (active) = passive

δext (in, passive) = active

in

δext (in, active) = active

λ(active)

{ send output, ack and done signals

}

ta(passive) = INFINITY

ta(active) = receiving\_time

**PQ\_System:**

**High Priority Queue & Low Priority Queue**

S = {passive, active}

X = {in}

Y = {out}

in

in

in

out

out

δint (active) = passive

δext (in, passive) = active

δext (in, active) = active

λ(active)

{ send the popped out element from front

}

ta(passive) = INFINITY

ta(active) = sending time

**Classifier:**

State Variables:

sigma = INFINITY, phase = Passive;

packetNum = 0; //the packet sequence number

totalPacketNumber = 0 //the total packet number that should be sent

sending = false; //true: sending packet, false: waiting for acknowledgement

ack =false; //true: expected acknowledgement is received, false: opposite

Formal specification:

X = {packet\_in\_n, ack }

Y = {pkt\_high\_out, pkt\_low\_out}

S = {{phase, sigma, packetNum, sending, ack}}

δext ( packetNum, sending, ack, e, x)

{ case phase

passive:

if x is from packet\_in\_n and phase

totalPacketNum = controlIn; //get input

packetNum = 1;

ack = false;

sending = true;

sigma = sending\_time; //transit to the state of sending packet

phase = active;

else

; //It should not be here. Input is ignored.

active:

if x is from ack

ack = true;

sending = false;

sigma = 0; //trigger an internal function immediately

}

else

; //It should not be here. Input is ignored.

}

δint (packetNum, sending, ack, e, x)

{ case phase

active:

if (ack) //expected acknowledgement is received

{ if (packetNum < totalPacketNum) //send next packet

{ packetNum ++

sending = true;

ack = false;

sigma = sending\_time;

}

else // all packets have been sent out successfully

{ phase = passive; //change back to initial passive state

sigma = INFINITY;

}

}

else if (sending ) //change to waiting mode after sending a packet

{ sending = false;

sigma = timeout;

}

else //time out, re-send the previous packet

{ sending = true;

sigma = sending\_time;

}

passive: /\*Never happens\*/

}

λ(active & sending)

{ send *packetNum* to the port *pkt\_high\_out* //packet sequence number

Send packetNum+1 to the port pkt\_low\_out

}

The formal specifications <X, Y, D, {Mi}, {Ii}, {Zij}, SELECT > for the coupled model PQ\_System and Toppq Simulator are defined as follows:

**PQ\_System:**

X = {pkt\_inch, pkt\_incl};

Y = {pkt\_high\_receiver, pkt\_low\_receiver};

D = {High\_Priority\_Queue, Low\_Priority\_Queue};

I(High\_Priority\_Queue) = self;

I(Low\_Priority\_Queue) = self;

Z(High\_Priority\_Queue) = self;

Z(Low\_Priority\_Queue) = self;

SELECT:({High\_Priority\_Queue, Low\_Priority\_Queue }) = High\_Priority\_Queue;

**TopPq Simulator:**

X = {packet\_in\_n};

Y = {packet\_outh\_n, packet\_outl\_n, ack};

D = {classifier, PQ\_System, receiverH, receiverL};

I(classifier) = { PQ\_System, self};

I(PQ\_System) = { receiverH, receiverL};

I(receiveH) = {self};

I(receiveL) = {self};

Z(classifier) = PQ\_System; Z(classifier) = self;

Z(PQ\_System) = classifier; Z(PQ\_System) = receiverH, receiverL;

Z(receiverH) = self;

Z(receiverL) = self;

**Test Strategies**

The atomic models and coupled models will be tested using the “black box” testing method. Test cases are created by adding different combinations of inputs to the event file (*.ev*), run the simulation and check whether the outputs in the output file (*.out*) are what we expected.

**Part III**

In order to verify the atomic models and coupled models, test cases are created to test these models.

**Test Cases and Execution Analysis**

**Atomic Model** **receiver (ReceiverH & ReceiverL):**

This model has two receivers to receive both high priority and low priority packets for further processing. The below information is for both of them in common.

The input of the receiver is supposed to a positive integer. The digits are assumed to be the packet data. The output is generated after a fixed time duration (e.g. 10 time units) when the input is received. The receiver should only work with one packet at a time. If a new packet arrives while the receiver is processing a packet, the older packet should be discarded. In the test cases, if the time duration between two consecutive inputs is less or equal than the receiver *receiving\_time*, the former input should be discarded, and no output will be generated for that input. It also generates done\_\* for the queues to the coupled model. The *receiverh.ev(receiverl.eh)* file is created as follows. It contains normal events and consecutive events with time duration less than or equal to the *receiving\_time* (i.e. 10 time units) of the receiver.

00:00:10:000 pkt\_low\_rec 1

00:00:30:000 pkt\_low\_rec 2

**00:00:45:000 pkt\_low\_rec 3**

00:00:52:000 pkt\_low\_rec 3

**00:01:25:000 pkt\_low\_rec 4**

00:01:35:000 pkt\_low\_rec 4

00:02:00:000 pkt\_low\_rec 5

In the above ev file, the two highlighted events doesn’t occur as the next packet at 45 seconds and 1 minute 25 seconds comes earlier and hence will be dropped. Below is the output file and hence the atomic model works as expected.

The output file *receiverH.out* shows the expected results.

00:00:10:000 pkt\_low\_rec 1

00:00:30:000 pkt\_low\_rec 2

00:00:45:000 pkt\_low\_rec 3

00:00:52:000 pkt\_low\_rec 3

00:01:25:000 pkt\_low\_rec 4

00:01:35:000 pkt\_low\_rec 4

00:02:00:000 pkt\_low\_rec 5

**Atomic Model** **High\_Priority\_Queue or Low\_Priority\_Queue:**

The input of the Queue is supposed to a positive integer. The input is sent into the queue only if the queue has empty space for the new element. There are separate queues for High priority packets and low priority packets. Once a packet is popped out from the front, a done input comes from the receiver to indicate that new packet can enter the queue. The time it takes is preparation time of 10sec.

The *High\_priority\_queue.ev* file is created as follows.

00:00:10:00 pkt\_inch 1

00:00:20:00 done\_hi 1

00:00:35:00 pkt\_inch 2

00:00:45:00 done\_hi 1

00:01:00:00 pkt\_inch 3

00:01:10:00 done\_hi 1

00:01:25:00 pkt\_inch 4

00:01:35:00 done\_hi 1

00:02:50:00 pkt\_inch 5

00:03:00:00 done\_hi 1

00:03:15:00 pkt\_inch 6

00:03:25:00 done\_hi 1

The outputs are generated after a done signal is seen. The following is an example of the output file *High\_Priority\_Queue.out. Similar is the case with Low\_Priority\_Queue. Hence the model works well.*

00:00:20:000 pkt\_high\_receiver 1

00:00:55:000 pkt\_high\_receiver 2

00:01:20:000 pkt\_high\_receiver 3

00:01:45:000 pkt\_high\_receiver 4

00:03:10:000 pkt\_high\_receiver 5

00:03:35:000 pkt\_high\_receiver 6

**Atomic Model Classifier:**

Classifier model accepts the inputs from the packet\_in\_n port and an ack signal from the receiver. **The input packet count is always divided into High priority packet number and a low priority packet number for easiness**.

The *classifier.ev* file is created as follows. It contains normal events and an illegal event.

00:00:10:000 packet\_in\_n 7

00:00:30:000 ack\_h 1

00:00:50:000 ack\_h 2

***00:00:55:000 ack\_h 3***

00:01:10:000 ack\_h 4

00:01:25:000 ack\_h 5

00:01:40:000 ack\_h 6

In the output file *classifier.out,* the event with ***bold and italic*** font is an illegal event that should be ignored. The Event (00:00:55:00 ack\_h 3) simulates a lost packet. pkt\_out\_l=3 and pkt\_out\_h = 4 are lost. The simulator works as expected.

00:00:20:000 pkt\_out\_l 1

00:00:20:000 pkt\_out\_h 2

00:00:40:000 pkt\_out\_l 2

00:00:40:000 pkt\_out\_h 3

00:01:05:000 pkt\_out\_l 4

00:01:05:000 pkt\_out\_h 5

00:01:20:000 pkt\_out\_l 5

00:01:20:000 pkt\_out\_h 6

00:01:35:000 pkt\_out\_l 6

00:01:35:000 pkt\_out\_h 7

**Coupled Model PQ\_System:**

The coupled model **PQ\_System** consists of two queue models. The two queues work independently. The *pq\_system.ev* is created as follows. Both the high and low outputs are given simultaneously.

00:00:10:000 pkt\_inch 1

00:00:20:000 done\_hi 1

00:00:10:000 pkt\_incl 2

00:00:20:000 done\_lo 1

00:00:40:000 pkt\_inch 3

00:00:50:000 done\_hi 1

00:00:40:000 pkt\_incl 4

00:00:50:000 done\_lo 1

Below is the output of the PQ\_system coupled model. And the simulator works as expected.

00:00:20:000 pkt\_high\_receiver 1

00:00:20:000 pkt\_low\_receiver 2

00:01:00:000 pkt\_high\_receiver 3

00:01:00:000 pkt\_low\_receiver 4

**Coupled Model Top PQ Simulator:**

The coupled model PQ Simulator is the top model which integrates atomic models Classifier, 2 receivers and coupled model PQ\_System. The packets are taken by the classifier and are classified into High and Low priority packets and are sent to the respective queues which are destined to the respective receivers. Acknowledgement from the receiver is received by the classifier to service th next packet.

The input of the top model is just *packet\_in\_n*, a positive integer indicating the number of packets needs sending in a session, where each packet is sub divided into high and low packets for easiness in this model. The output packet\_outh\_n and packet\_outl\_n are the outputs of the simulator.

The toppq*.ev* is very simple, just one line. It requires sending 20 packets in a session.

00:00:10:00 packet\_in\_n 10

The following is an example of the *toppq.out*.

00:00:40:000 packet\_outh\_n 2

00:00:40:000 packet\_outl\_n 1

00:01:10:000 packet\_outh\_n 2

00:01:10:000 packet\_outl\_n 1

00:01:40:000 packet\_outh\_n 3

00:01:40:000 packet\_outl\_n 2

00:02:10:000 packet\_outh\_n 4

00:02:10:000 packet\_outl\_n 3

00:02:40:000 packet\_outh\_n 5

00:02:40:000 packet\_outl\_n 4

00:03:10:000 packet\_outh\_n 6

00:03:10:000 packet\_outl\_n 5

00:03:40:000 packet\_outh\_n 7

00:03:40:000 packet\_outl\_n 6

00:04:10:000 packet\_outh\_n 8

00:04:10:000 packet\_outl\_n 7

00:04:40:000 packet\_outh\_n 9

00:04:40:000 packet\_outl\_n 8

00:05:10:000 packet\_outh\_n 10

00:05:10:000 packet\_outl\_n 9

00:05:30:000 packet\_outh\_n 11

00:05:30:000 packet\_outl\_n 10

The PQ Simulator model simulates the priority queueing Protocol and generates the expected results. The behaviors and features of the classifier, PQ\_system and receiver are simulated by the respective models. The data of the packet is simplified as a packet sequence number in this model and also the low and high priority packets are deduced from a packet for easiness. The testing cases verify the specifications of models. The PQ Simulator works exactly as we expected according to the specifications.