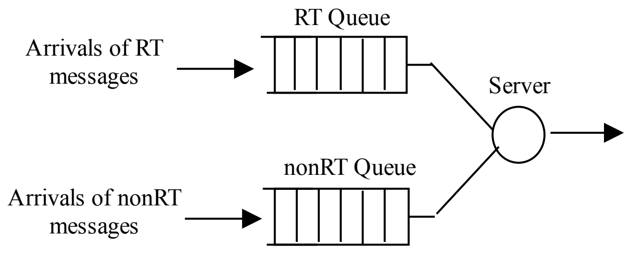
Simulation project
Task 1- Hand simulation

Problem description

We will consider a simple IoT system where a large number of sensors send messages to a server. Each message contains information collected by a sensor. We assume that the server is near the sensors so that the propagation delay from a sensor to the server is negligible. The server processes the messages and issues commands to actuators. We assume that there are two types of messages, real-time (RT) and non real-time (nonRT). RT messages have to be processed as fast as possible since they represent tasks that need to be executed in real time. NonRT messages represent non real-time tasks and therefore they are not time constrained. The server maintains two queues, an RT queue and a nonRT queue, as shown below.



The RT queue has a preemptive priority over the nonRT queue. That is:

1. Each time the server completes a service, i.e., processing a message, it checks the RT queue to see if there are any messages waiting. If there is a message waiting, then it starts processing it.
2. If the RT queue is empty, the server checks the nonRT queue.
    a. If there is a message waiting, it starts processing it.
    b. If there are no messages waiting, the server becomes idle.
3. If during the time the server is processing a nonRT message, an RT message arrives, the server interrupts the processing of the nonRT message and starts processing the RT message. Upon completion of processing the RT message, the server selects the next message by going back to step 1.
4. When the server processes an interrupted nonRT message, it starts processing it from where it stopped when it was interrupted. (This policy is called pre-emptive resume and it is common in CPU scheduling.)

Project description

You will develop an event-based simulation model of the two-queueuing system at the server, with a view to calculating the 95th percentile of the *response time* of RT and nonRT messages. The response time is defined as the time elapsing from the time an RT (nonRT) message joins its queue to the moment it is fully processed by the server and departs from the server.

For pedagogical purposes, the simulation project is broken in the following three tasks:

1. *Task 1 - Hand simulation:* The objective is to make sure that you understand the events, the event clocks, and how the simulation advances from one event to another.
2. *Task 2 - Code basic simulation:* The objective of this task is to code of the simulation logic and to make sure that the program works correctly by carefully checking the output.
3. *Task 3 - Statistical estimation*: The objective is to embellish your simulation with additional code so that to estimate statistically (with confidence intervals) the response time of the RT and nonRT messages.

Grading of the simulation project.

Each task will be graded from 0 to 100 and then the grades will be combined to a single project grade using the following weights:

- 10% for task 1
- 40% for task 2
- 50% for task 3

The simulation structure

The state of the simulation model can be described by the following basic parameters:

- *Number of RT messages in the RT queue,* n_{RT}
- *Number of nonRT messages in the nonRT queue,* n_{nonRT}
- *State of the server s: idle (s=0), serving an RT message (s=1), serving a nonRT message (s=2).*

There are other parameters too, but they are not critical to the development of the simulation model. The simulation is based on tracking the events that change the state of the system through time. These events are:

1. *An RT arrival occurs*
2. *A nonRT arrival occurs*
3. *The processing time of an RT or nonRT message is completed*

Note, that the occurrence of one event may trigger the occurrence of one or more events. Below, we discuss what happens when an event occurs.

1. An RT arrival occurs

The message joins the queue, that is, the number n_{RT} currently in the queue is increased by 1. No further action is required, if there are other messages waiting in the queue. However, if this message finds the queue empty, then it may be possible that it can start its processing immediately. Check the status of the server, and schedule a new service time if it is idle. If it is busy serving an RT message then no further action is required. If it is busy serving a nonRT message then the service is interrupted and the server starts processing the RT message. The interrupted nonRT message returns to the top of the nonRT queue and it will be processed at a later time per the scheduling algorithm. Its processing will start from where it stopped when it was pre-empted.

Before leaving this part of the logic, you need to generate the next arrival time of an RT message.

2. A nonRT arrival occurs

The message joins the queue, that is, the number n_{nonRT} currently in the queue is increased by 1. No further action is required, if there are other messages waiting in the queue. On the other hand, if this message finds the queue empty, then it may be possible that it can start its processing immediately. Check the status of the server, and if idle schedule a new service time. No further action is required if it is busy serving an RT or a nonRT message.

Before leaving this part of the logic, you need to generate the next arrival time of an nonRT message.

3. A service completion occurs

In this case, the server runs the scheduler to determine the next message to process. Specifically, it checks first the RT queue. If the number of messages n_{RT} in the queue is greater than 0, then at least an RT message is waiting. The server starts processing the first message at the top of the RT queue and n_{RT} is reduced by one. Schedule the time of service completion. If $n_{RT} = 0$, then it checks the nonRT queue. If the number of nonRT messages n_{RT} currently in the queue is greater than 0, then it starts processing the first message at the top of the nonRT queue. Schedule the time of service completion and reduce n_{nonRT} by 1. If a nonRT message has been pre-empted before, the service time is the left-over service at the time when the message was pre-empted.

The simulation logic

Each of the three events is associated with a *clock* that gives the time of completion of the event in the future. Let these clocks be: *RTCL*, *nonRTCL*, and *SCL* for the event of an RT arrival, a nonRT arrival, and a service completion correspondingly. The event clocks are used to decide which event will be executed next. In addition, we use a *master clock* (*MC*), which tells us the current time. These clocks are just variables in your program and not the computer's clock!

Main logic – chose the next event

Each time an event has been serviced, the simulation logic checks all the event clocks scheduled to occur in the future. It then advances to the event with the smallest clock value. Depending on the event, it takes the following action.

1. Arrival of an RT message

- Set *MC* to *RTCL*
- Message joins the RT queue. Increase $n_{RT}$by 1.
- Generate the next inter-arrival time *IAT* of an RT message and set *RTCL=MC+IAT.*
- If n_{RT}=1, then check if server is idle.
    - If yes, message begins processing at the server.

  - Determine the length of the service time *ST* and set *SCL=MC+ST*.
  - Decrease n_{RT} by 1.
  - Set s = 1. Recall that s is the state of the server: s=0 (idle), s = 1 (serving an RT message, s=2 (serving a nonRT message).
- If not, check if the server is busy serving a nonRT message. If yes, the nonRT message is preempted and the RT message begins processing.
  - If the remaining service time of the pre-empted nonRT message is not zero, then:
    - Store the remaining service time.
    - Return the pre-empted message to the nonRT queue and increase n_{nonRT} by 1.
  - If the remaining service time of the pre-empted nonRT message is zero, then:
    - The nonRT message has completed its service and it departs from the system.

    Note: this condition can only happen when the clocks are integers, which is what we assume in tasks 1 and 2.1. In tasks 2.2 and 3 you will define the clocks as real variables, which means that two event clocks can never be the same.
  - Determine the length of the service time *ST* and set *SCL=MC+ST*.
  - Decrease n_{RT} by 1.
  - Set s = 1
- Go back to the beginning to locate the next event.

2. Arrival of a nonRT message
- Set *MC* to *nonRTCL*
- Message joins the nonRT queue. Increase $n_{nonRT}$by 1
- Generate the next inter-arrival time *IAT* of a nonRT message and set *nonRTCL = MC+IAT*.
- If n_{nonRT}=1, then check if server is idle. If yes, message begins processing at the server.
  - Determine the length of the service time *ST* and set *SCL=MC+ST*.
  - Set s=2.
- Decrease n_{nonRT} by 1.
- Go back to the beginning to locate the next event.

3. Service completion
- Set *MC* to *SCL.*
- Check the RT queue.
  - If there is an RT message waiting in the queue, calculate its service time *ST* and set *SCL=MC+ST.*
  - Set s=1.
  - Decrease n_{RT} by 1.
- If RT queue is empty, check the nonRT queue.
  - If there is an nonRT message waiting in the queue calculate its service time *ST* and set *SCL=MC+ST.*
  - Set s=2.
  - Decrease n_{nonRT} by 1.
- If both queues are empty, the server becomes idle. Set s=0.
- Go back to the beginning to locate the next event.

Hand simulation

In order to understand how the simulation logic works, we start with the hand simulation given in the table below. For this, we will make the following simplifying assumptions:

- The inter-arrival time of RT messages is constant = 10
- The inter-arrival time of nonR is constant = 5
- The service time of an RT message is constant = 2
- The service time of a nonRT message is constant = 4

In order to start the simulation, we will make an assumption as to the state of the simulation at time *MC*=0. For the hand simulation given below, we will assume that: $RTCL$ = 3, *nonRTCL*=5, n_{RT}=0, n_{nonRT}=0, *s*=2, *SCL*=4. The set of the starting assumptions is known as the *initial conditions*, and as will be seen later on, the final results that we will obtain from the simulation do not depend on the initial conditions.

Note that it is possible that an arrival event may occur at the same time as a service completion event. In this case, first execute the arrival event and then the service completion event. This problem of simultaneously occurring events will go away in the subsequent two extensions of the simulation because the clocks will be all defined as real variables.

Deliverables for task 1

Task 1.1:
Do the hand simulation until *MC*=50.

Task 1.2:
Do the hand simulation until *MC*=20, but now reverse the input parameters as follows. The inter-arrival time of RT messages is constant = 5, the inter-arrival time of nonR is constant = 10, the service time of an RT message is constant = 4, and the service time of a nonRT message is constant = 2.

Use the same initial conditions.

What to submit
Submit your results to Moodle in two separate tables similar to the one below. Prepare your tables using a word processing software. Alternatively, you can hand write them, scan them and upload them to Moodle.

Make sure your hand simulations are correct because this will be the basis of your remaining simulation tasks.

Grading

- 50 points for task 1.1
- 50 points for task 1.2

Table 1: Task 1.1

MC	$RTCL$	$nonRTCL$	n_{RT}	n_{nonRT}	SCL	**Server status, s**	**pre-empted service time**
0	3	5	0	0	4	2	
3	13	5	0	1	5	1	s=1
5	13	10	0	2	5	1	s=1
5	13	10	0	1	6	2	
6	13	10	0	0	10	2	
10	13	15	0	1	10	2	
10	13	15	0	0	14	2	
13	23	15	0	1	15	1	s=1

Table 2: Task 1.2

MC	$RTCL$	$nonRTCL$	n_{RT}	n_{nonRT}	SCL	**Server status, s**	**pre-empted service time**
0	3	5	0	0	4	2	
3	8	5	0	1	7	1	s=1
5	8	15	0	2	7	1	s=1
7	8	15	0	1	8	2	
8	13	15	0	1	12	1	
12	13	15	0	0	14	2	