Final Project for DESI (2021) (first draft)

e-meeting for final project presentation: to be arranged

Final project due date: to be scheduled

Define and implement a Factory DES System

The factory model consists of the following equipment and modules:

1. A part release module: Generates and releases parts of type *A* and *B* to the factory, according to relative probabilities (frequencies) p_a and p_b specified by the user. Therefore, ratio $p_a/(p_a + p_b)$ is the probability for generating product *A* and $p_b/(p_a + p_b)$ is for product *B*. The relative probabilities are p_a=30, p_b= 70. The interarrival times between two successive part releases are Exponentially (μ = 2) distributed.

2. A part washing workstation consists of one machine *W*, used to clean the entered parts. The washing time is stochastically set by the machine and follows Normal distribution (μ = 1, σ = 0.5) with a lower bound 0.1 and a upper bound is 5.0. A capacitied buffer Q_w with capacity 10 is installed in the workstation for *W* to queue the parts, which are FCFSed.

3. A press workstation consists of three parallel press machines P_1, P_2, and P_3. They are of the same type and used to shape the parts. Different types of parts take different processing times, whose distribution types are specified in individual itinerary. Each machine has its own buffer Q_{Pi} to hold unprocessed parts and uses FCFS policy to process them. Buffers are capacitied whose sizes Q_{P1}, Q_{P2}, and Q_{P3} are 8, 7, and 6, respectively.

4. An assembly workstation consists of two assembly machines Y_1 and Y_2. Y_1 and Y_2 are of the same type and used to assemble parts. The processing times are stochastically determined by the parts and their distribution types are specified in each distinct itinerary. A prioritized buffer Q_y with capacity limit 9 is installed to store parts for these machines. Priority policy is used in Q_y to select parts for assembling for a free machine. Parts of type *A* have a higher priority than *B*. For example, if the types of the queued clients are (*B*, *B*, *A*, *A*, *B*, *A*, *B*), the third client is dequeued for service, since it is the first part with the type A. If the type list of clients is (*B*, *B*, *B*), the first client of type B is dequeued.

5. Machines have additional states: BreakDown (or Repairing). The time-to-failure and time-to-repaired of all machines are supposed to be Exponential (μ = 500) and Exponential (μ =

10).

The process plans (itineraries) of part *A* and *B* are:

1. Part A: $W \rightarrow P_1$, P_2, or P_3 (Exponential ($\mu = 8$)) $\rightarrow Y_1$ or Y_2 (Exponential ($\mu = 5.5$))
2. Part B: $W \rightarrow P_1$, P_2, or P_3 (Exponential ($\mu = 6$)) $\rightarrow W \rightarrow Y_1$ or Y_2 (Exponential ($\mu = 4.5$))

The cease time of part release is 2000 and simulation runs to the last event. Note that, once the processing of a part is finished by a machine and the part can not enter the next workstation for successive processing, the finished part remains on the machine. Therefore, the state of the machine turns into *blocked* from busy, while the state of the part turns to *dwell*, form *beingServied*. Your challenges are:

(1) deriving from the generic Server, define a Machine class that overrides the inherited functions and adding a MachineBreakDown and a MachineRepaired discrete events and design their ProcessEvent() functions,

(2) designing an itinerary class that describes a sequence of service nodes associated with service time generators, if the service time is determined by the client that take the itinerary,

(3) adding a priority weight to the itinerary class so the clients take different itineraries will have different priorities if they are queued in a prioritized queue,

(4) extending your queue class to be able to deal with capacity limited queues that will not receive client when it is full,

(5) extending your queue class or deriving it to be able to perform as a prioritized queue that sorts the sequence of clients according to their priorities, not the entering times,

(5) enhancing the client generator to serve as a part releaser that is able to release parts taking different itineraries to simulate a product mix manufacturing.

Programming Help/Instruction for enhancing midterm project for the final project

Enhance your timed queue class to a capacitied FCFS queue, having a property of queue capacity that can be specified by the user. Enhance its method of adding a client to return false to the caller (ServiceNode) when its capacity is full.

In the current implementation, the function of the service node that escorts a client

(whose service is completed by a server) will be failed when the client has a next service node to visit and the node refuses to receive any client. In this case the server turns “blocked” and the client occupies the server. Blocked servers are then recorded in the service node for service resuming. The service completed function of a blocked server will be resumed when the occupied consumer can be escorted to the next node.

Enhance the data structure and functionalities of the service node class. The function receiving a client will return “false” to the caller, when all the servers within this node are not free (busy or blocked) and all queues do not have any vacancy left. Therefore, the uncompleted service done operation of the blocked server should be resumed when the node has a vacancy. Note that a vacancy happens when the node successfully escorts a client. Thus, the node must have a server queue (e.g., deferredServerQueue) to store the deferred servers that fail to escort the client. In addition, you need to add a server parameter to the function that receives a client. When the client is prohibited to enter the node, its blocked server is recorded. You need to enhance the function of the service node that escorts a client to invoke the complete service function of the head deferred server in the queue when a client is successfully escorted by the node and a vacancy is available for the deferred servers, if the queue is not null.

Create an *itinerary* or *route* class that loops through a list of service nodes. Enhance the client class to have an object of the itinerary or route class and an index that indicating the currently visited service node. For each service node to be visited in an itinerary, a reference of a random variate generator is specified for generating the servicing time in the visit. Therefore, in your server class, the function that serves a consumer will not use the service time generator of the server; instead, the one embedded in the itinerary whose reference is not null. If the time generator is null in the itinerary, the service time is determined by the server.

The itinerary class should have a priority weight indicating the processing priority of the client that takes the itinerary. In a prioritized queue, the sequence of queued clients follows the orders of their priorities. Therefore, the client class should post its itinerary property and itinerary class should reveal its priority weight. A prioritized queue will insert the newly entered client into the client queue in descending orders of their priorities, not the entering times.

Update your class diagram and pack it up with your source code. Prepare the UIs for your final project in advance. Exercise the enhanced functionalities. Start updates of the

computation flows of the two event processing functions.

In your final project, the generic server has three distinct states: *Free*, *Busy*, and *Blocked*. The *Blocked* state is resulted from the capacity full of the next service node for the served client. The class of server might be used to instantiate a bank teller, a machine, a bus, and so on. However, the generic server does not have a *break down* state, while objects of machines have. Therefore, your Machine class inheriting from server should have extra data fields relating to properties of *mean time to failure* and *mean time to repair* (associated with distribution type) that can be specified by the user.

Here we suppose the repairing resource is infinite. Once a machine is down, its status turns into *Breakdown* right away and then turns to *Free* after a stochastic repairing time. Therefore, you need to add two derived discrete events to simulate machine breakdown and repaired.

When the simulation is reset, in addition to adding the first arrival event, each machine should generate its first breakdown event and add to the event queue. The overridden processEvent() function of the breakdown event should ask the server (call function provided by the server) to move the clients under servicing to other servers (or queues), if the breakdown happens in busy state. The server in turn asks help from the service node (via function call) to conduct this task, and then generates a repaired event to the event queue.

The overridden processEvent() function of the repaired event will ask the server (machine) to reopen service. If dedicated queue is used by the server, the server should ask the service node to empty the queue and redistribute the queued clients. Finally, the next breakdown event for the server (machine) should be generated and added to the event queue.

Extra Credits:

To illustrate the beauty of O-O design, interactive modeling capability should be provided to let user construct complicated simulation models and save models for retrieval. Moreover, graphics display of the model and simple animation will help let the user comprehend the structure of the model and the progress of the simulation. Graphics display can also help your debug your code of modeling and simulation. Follow the instruction videos, learn the programming techniques to enhance the applicability of your DES library.

Submission Requirements of the final project

Use object-oriented techniques to analyze, design, and implement the DES system for the designated factory. Submit your final project artifacts, and prepare your project presentation according to the following guidelines:

System Execution Video:

Make a video that shows the execution of your system. If interactive model construction capability is available, record a video to show the construction steps.

System Presentation and Demonstration Slides and Video:

1. Prepare no more than 12 presentation slides to illustrate the work of your project.
2. Make a 10-minute presentation video using the slides your prepared. In the video, demo the modeling and simulation capabilities of your system and run the factory model to show the results. You can run your system execution video or do real-time demo on your system.
3. If e-meeting were held, you will have at most 10 min. to present your materials and another 3 mins to demonstrate your system.

Material Submission:

(1) Source files of your project.

(2) Saved model files of the target factory, if the model can be saved as file.

(3) Application demo video file (or the cloud link).

(4) Presentation slides file.

(5) Presentation video file (or the cloud link).

Be sure to wrap up your presentation file, video file (or cloud link), and all source code of your application within a folder named after your student ID. Archive the folder to a compressed file. Submit it to the course web site.