David Perry CS543 HW#3 Nov 18, 2014

Priority with Preemption:

In this experiment I was asked to create a M/M/1 queue that had two different types of customers. The system contains high priority customers (type 1) and low priority customers (type 2). In this simulation all existing type 1 customers are served before any existing type 2 customers and interrupt the service of a type 2 customer . The main implementation details of using CSIM to create the simulation are as follows:

- As with all CSIM simulations a sim function must exist and contain a call to create to make it become a process
- A facility is declared. This structure represents the queue. The set_servicefunc function is used to change the service discipline of the facility to preempt.
- An event is created to signal when the simulation is completed.
- Two functions are called that call create to become processes. These two processes are responsible for creating low-priority and high-priority processes respectively at a rate that corresponds to a exponential random number.
- Each low-priority and high-priority process call use on the facility. This places the processes in a queue and serves them with preemption.
- After each process holds the facility for a exponential random amount of time the process decrements the number of processes served counter and checks if it is the last to be served. If so, the process signals the event created earlier, sim resumes execution, and simulation ends.

Simulation Results:

The results for the simulation can be found below. As can be seen the simulated results were extremely close to the theoretical results in part 1 and from the original simulation implemented in homework 2. All runs were done with 1000 customers and varying seed values.

Theoretical Simulation Calculations:

E(N_1) = (p_1/(1 -p_1)
E(W_1) = 1/(μ(1-p_1))

E(N_2) = p_2/((1-p_1)(1-p_1-p_2)
E(W_2) = 1/(μ(1-p_1)(1-p_1-p_2)

Example Calculation:

p_1 = .2
p_2 = .5
μ = 1

E(N_1) = (.2/(1 -.2) = .25
E(W_1) = 1/(1(1-.2)) = 1.25

E(N_2) = .5/((1-.2)(1-.2-,5) = 2.0833

$E(W_2) = 1/(1(1-.2)(1-.2-.5) = 4.166$

Home Work 2 Simulation Results:

$p_1 = .2$
$p_2 = .5$
$\mu = 1$

$E(N_1)$	$E(W_1)$	$E(N_2)$	$E(W_2)$
0.253784	1.20722	2.531767	4.71796
0.267171	1.19377	2.089839	4.42790
0.281736	1.30822	2.465020	4.71225
0.242819	1.17138	1.832631	3.78892
0.266560	1.34248	2.179017	4.39502

Averaged Simulation Results:

$E(N_1)$	$E(W_1)$	$E(N_2)$	$E(W_2)$
.263414	1.244614	2.2196548	4.40841

Home Work 3 Simulation Results:

$p_1 = .2$
$p_2 = .5$
$\mu = 1$

$E(N_1)$	$E(W_1)$	$E(N_2)$	$E(W_2)$
0.248453	1.332007	1.687095	3.517439
0.228199	1.211578	2.037775	4.053346
0.243636	1.269182	2.118490	4.060916
0.218062	1.134705	1.806648	3.760501
0.243250	1.110081	2.145048	4.235109

Averaged Simulation Results:

$E(N_1)$	$E(W_1)$	$E(N_2)$	$E(W_2)$
.23632	1.2115106	1.9590112	3.9254622

David Perry CS543 HW#3 Nov 18, 2014

Runtime Results:

The CSIM simulation was considerably faster than the simulation I wrote for homework. This is surprising as the CSIM program should have suffered a performance penalty due to the context switching overhead involved with processes. However, it is likely my simulation is slower due to not carefully coding for performance and only considering correctness. The runtime results for both simulations can be found below:

Homework 2 Runtime Results:

Real	User	Sys
0.855s	0.806s	0.038s
0.855s	0.828s	0.022s
0.858s	0.831s	0.021s
0.855s	0.829s	0.022s
0.864s	0.835s	0.022s

Averaged Runtime Results:

.8574	.8258	.025

Homework 3 Runtime Results:

Real	User	Sys
0.020s	0.012s	0.006s
0.021s	0.012s	0.006s
0.021s	0.012s	0.006s
0.018s	0.012s	0.006s
0.020s	0.012s	0.007s

Averaged Runtime Results:

.02	.012	.0062

2.)

WIDGET XYZ, INC Simulation:

In this experiment I was asked to create a simulation of operations at company WIDGET XYZ, INC in CSIM. The features that need to be simulated are the production of widgets, incoming order requests, and packing. The main implementation details of using CSIM to create the simulation are as follows:

- First all the data structures required for the simulation were created. These include an event to signal simulation completion, a storage for keeping track of inventory and incoming order requests, and a facility to simulate the packing machine.
- The first process created is for production. This process simply loops until the simulation is over, holds for a uniform random amount of time, and adds 1 widget to the storage.
- Next an order dispatch process is created. This process loops until the simulation count is zero, holds for an exponential random amount of time, and calls the order function.
- The order function creates the order process. The number of orders currently waiting on the storage is checked to see if the order back log is full. If not, a uniform random number is used to determine how many widgets the order requires, the process requests that many widgets from the storage, and uses the packing machine. If so, the order is discarded and recorded.

Results:

I have placed all the results in the chart below. Each column corresponds to an individual part of the question. Each experiment was ran 5 times and the values were averaged. Each simulation was ran with 10000 number of served orders.

(A)	**(B)**	**(C)**	**(D1)**	**(D2)**
5175	0.825	8.83634	0.84968	381.23732
4828	0.826comments	8.82370	0.85018	379.94116
4927	0.825	8.79636	0.84856	380.61942
4684	0.827	8.79188	0.85172	380.33417
4894	0.826	8.81243	0.84876	380.67510

Averaged Results:

4901.6	.8258	8.812142	.84978	380.561434

David Perry CS543 HW#3 Nov 18, 2014

3.)

Traffic Light Simulation

In this experiment I was asked to create a simulation of a two lane highway that is being repaired. Due to the repair there is a segment of highway that is only one lane. In order for this to be safe there is a traffic light placed at each location where the one lane begins. The amount of time each light stays green is the subject of our experiment (what we are optimizing) but in between each green light both lights must be red for 55 seconds (to ensure that all traffic is able to pass through safely). The main implementation details of using CSIM to create the simulation are as follows:

- First all the data structures required for the simulation were created. Two events were created representing a traffic light turning green. Two facilities were created to simulate the cars waiting at the two traffic lights.
- A process called traffic controller is created. This process loops until the simulation is over, sets the one of the light events, holds for an exponential random amount of time, clears the event, holds for 55 seconds (red light), signals the other green light,and holds for another exponential random amount of time.
- Another two processes are created at the beginning of the simulation. These processes simulate the arrival of cars in both directions. The process loops until enough cars have passed, holds for an amount of time (secs per car), and calls the car function.
- The car function creates the car process. Each car process attempts to reserve their corresponding light. Once a car is able to reserve the facility it holds for 2 seconds (1 car passes every 2 seconds), releases the facility, and decrements the counter.

Optimizing the Parameters:

To optimize the parameters I changed my CSIM program to handle command line arguments and have two parameters for length of time for each light to be green. I then wrote a python script (opt.py) that ran all combinations of 1 to 300 for each parameter. All executions that crash are discarded (too small of a green light time means too many processes get queued at a facility). The script then gets the wait time for both facilities, averages their value, and replaces the best known value if the current execution is better.

Results:

The optimal times for my experiment were:

GREENAB: 35
GREENBA: 47