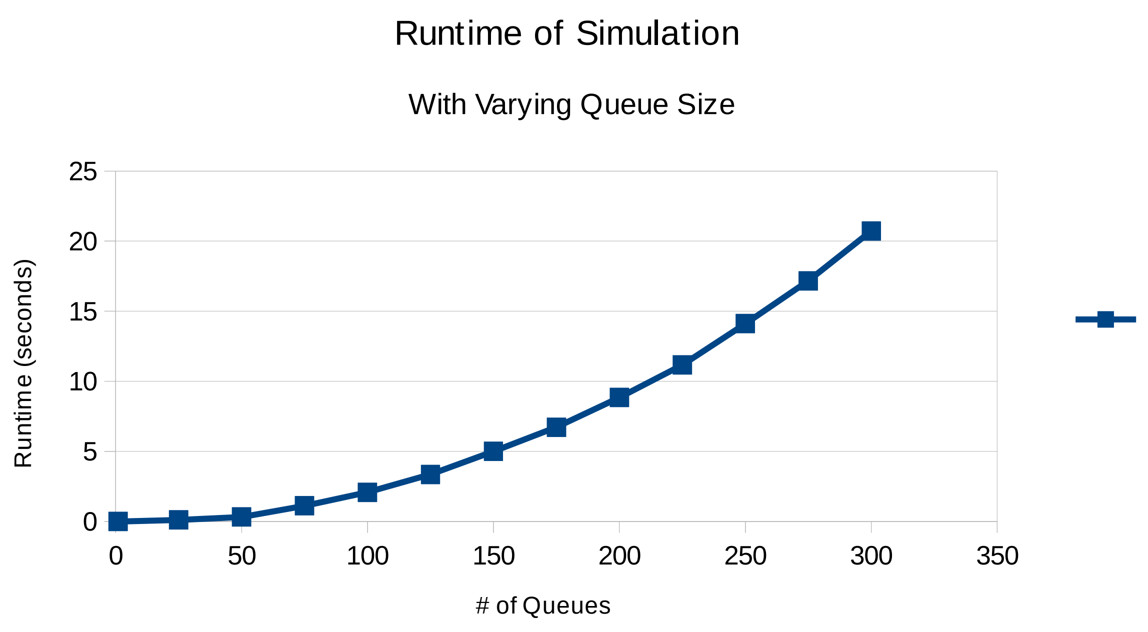
David Perry CS 543 HW#2 October 19. 2014

1.)



In this experiment I was asked to create a simulation of a N M/M/1 queue. The queue is a tandem queue meaning that as a customer exits queue i they arrive at queue i+1.This simulations required only slight modification to the standard M/M/1 queue that I implemented for homework 1. The most impactful implementation changes that had to be made were:

- Creating N queue structures. To accomplish this task the program creates an array of queues where the index indicates which queue.
- Creating a boolean indicating if the server is busy for N queues. This was also implemented as an array of integers.
- New arrivals must be scheduled when a customer leaves a (non N-1) queue. Since the queue is tandem when a customer exits queue i they enter queue i+1.
- Event records were modified to include an integer indicating the which queue the customer currently belongs to.
- The counter of how many customers have been served is only incremented when a customer leaves the N-1 queue.

Results:
The point of this simulation was to observe the runtime performance of the simulation rather than simulation data. According to the handout the complexity of the simulation process and number of queues impacts performance. However, since the simulation process is very simple runtime is mainly impacted by the number of queues. The impact of the number of queues on runtime can be seen above. The graph indicates that the runtime of the simulation grows exponentially as the number of queues increase.

David Perry CS 543 HW#2 October 19. 2014

2.)

Priority without 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 regardless of when they arrived. Again, this problem required modifications to the M/M1 queue simulation implemented in homework 1. The most important implementation changes that have to be made are:

- Create two queues to hold waiting customers. One queue for low priority customers and another for high priority customers.
- How customers are selected from the queue when a customer exits was changed so that high priority customers are selected first and low priority customers are only selected when there are no high priority customers.
- Average number of people in system and average time spent in system had to be calculated separately for each type of customer.

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. All runs were done with 1000 customers and varying seed values.

Part 2 of the results show how the system behaves when p_1 + p_2 = .9 for all values .1 to .8. The number of type 2 customers in the system appear to increase exponentially as their arrival rate increases. Type 2 customers does the same until it reaches a peak around p_2=.6. This is likely the case because type 2 and 1 customers are arriving at a very similar rate. Therefore, type 1 customers are arriving quickly but so are type 2. Therefore, type 2 customers typically have larger wait times.

Part 1

Theoretical Simulation Calculations:

E(N_1) = (p_1(1 + p_2))/(1 - p_1)
E(W_1) = (1 + p_2)/(μ(1 - p_1)

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

Example Calculation:

p_1 = .2
p_2 = .666

$\mu = 1$

$E(N_1) = (.2(1 + .5))/(1-.2) = .416$
$E(W_1) = (1 + .666)/(1(1-.2) = 2.083$

$E(N_2) = ((1 – .2(1 – .2 – .666)).666)/((1 – .2)(1 – .2- .666)) = 6.083$
$E(W_2) = ((1 – .2(1 – .2 - .666)))/(1(1 – .2)(1 - .2 - .666)) = 9.12497$

Simulation Results:

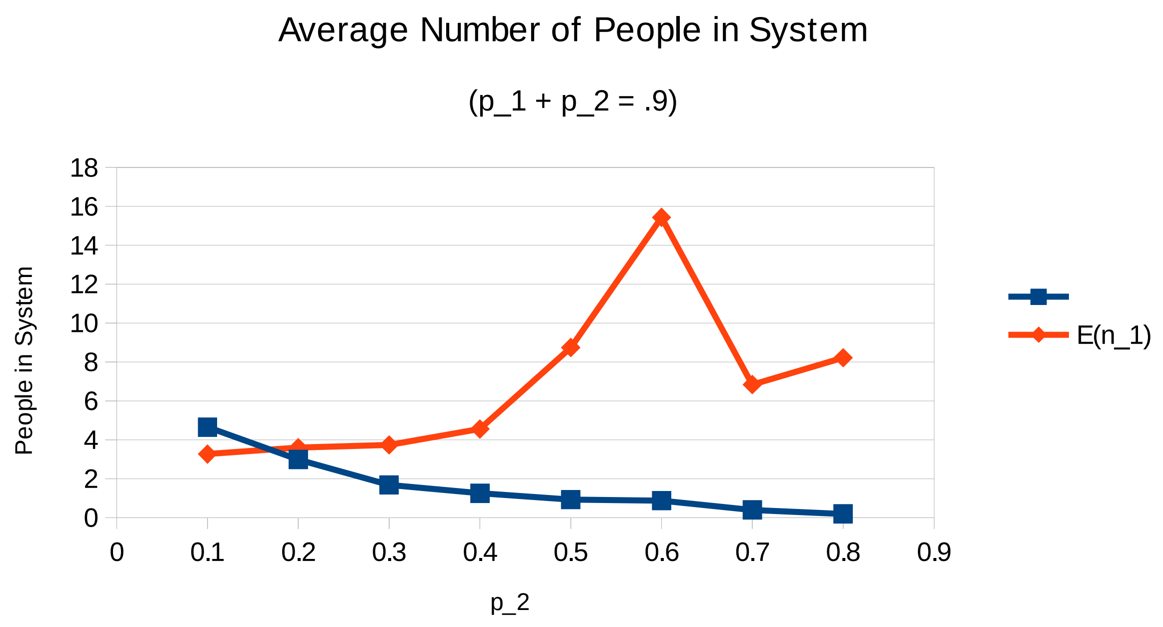
$p_1 = .2$
$p_2 = .5$
$\mu = 1$

E(N_1)	E(W_1)	E(N_2)	E(W_2)
.476138	2.38167	6.861168	9.72554
.403672	2.08369	5.372408	8.01578
.421613	2.06958	4.315947	6.70855
.462231	2.48255	6.960498	10.06218
.386894	1.93454	4.068261	6.06318

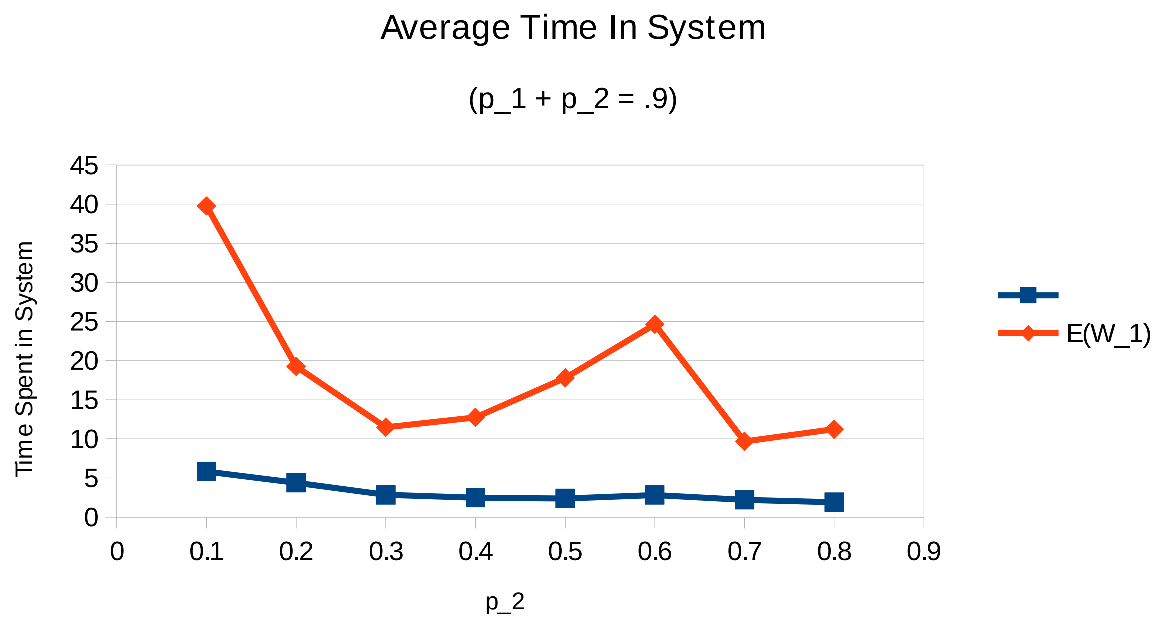
Averaged Simulation Results:

E(N_1)	E(W_1)	E(N_2)	E(W_2)
.4301096	2.190406	5.5156564	8.115046

Part 2:



Sorry, I couldn't get the graph's legend to be correct. Blue is E(n_1), Orange is E(n_2)



Sorry, I couldn't get the graph's legend to be correct. Blue is E(w_1), Orange is E(w_2)

David Perry CS 543 HW#2 October 19. 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 can interrupt the service of a type 2 customer . Again, this problem required modifications to the M/M1 queue simulation implemented in homework 1. The most important implementation changes that have to be made are:

- Create two queues to hold waiting customers. One queue for low priority customers and another for high priority customers.
- How customers are selected from the queue when a customer exits was changed so that high priority customers are selected first, any preempted type 2 customers are selected second and low priority customers are only selected when there are no high priority customers.
- A field to the events had to be added indicating how much time they had left. This is only calculated (and stored) when an event is preempted. This was calculated by looking at how much time the type 2 customer got served (before preemption) and subtracting it from the total amount of time it needed to be served.
- Exit events for preempted customers had to be canceled. To implement this my simulation numbers each event record with a unique number and a list of all canceled events is stored in an array. When an event is extracted from the heap it is checked to see if this event wasSorry, I couldn't get the graph's legend to be correct. Blue is E(n_1), Orange is E(n_2) canceled. If so, the event is discarded and another event is pulled from the heap.
- Average number of people in system and average time spent in system had to be calculated separately for each type of customer.

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. All runs were done with 1000 customers and varying seed values.

For the average number of people in the system the results were as expected for type 1 customers. As type customers arrived faster they spent more time in the system. For type 2 customers there was a large amount of variance. The number of type 2 customers stayed close to the same until p_2 was very large causing growth. This is because as type 2 customers arrive more frequently they must wait in the system for longer periods of time.

For time spent in the sytem the results were as expected for type 1 customers. As type 1 customers arrive quicker they have to wait longer. This makes sense as type 1 customers only have to wait on other type 1 customers. Type 2 customers spent the most time in the system when p_1 was large. This is because type 1 customers get to interrupt the service of type 2 resulting in more wait time.

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

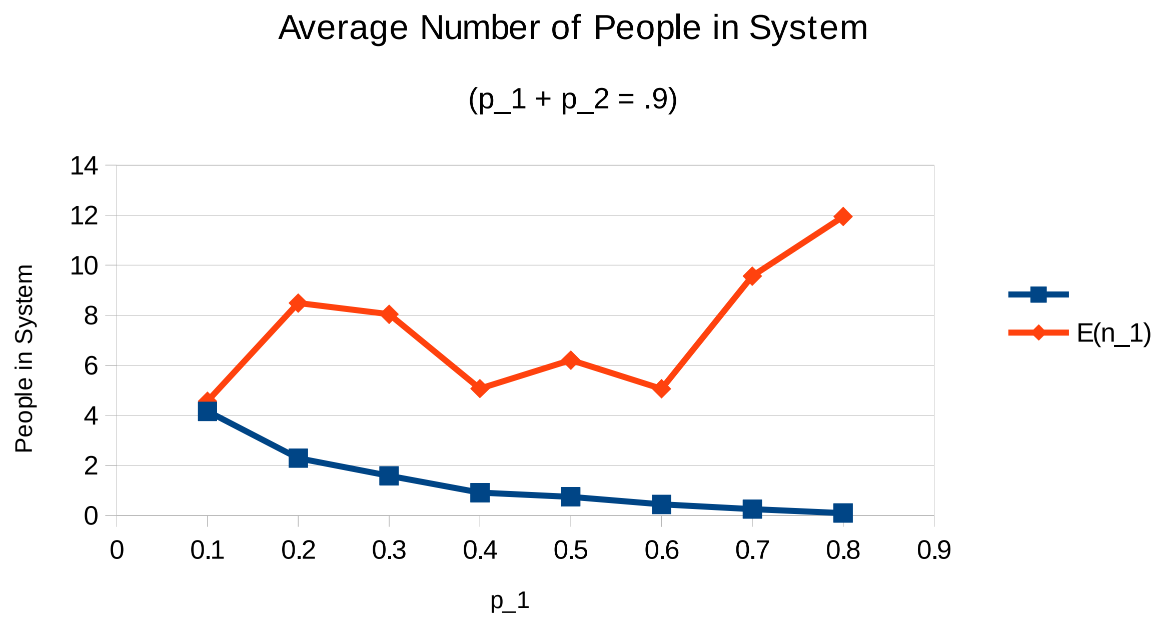
Simulation Results:

p_1 = .2
p_2 = .5
μ = 1

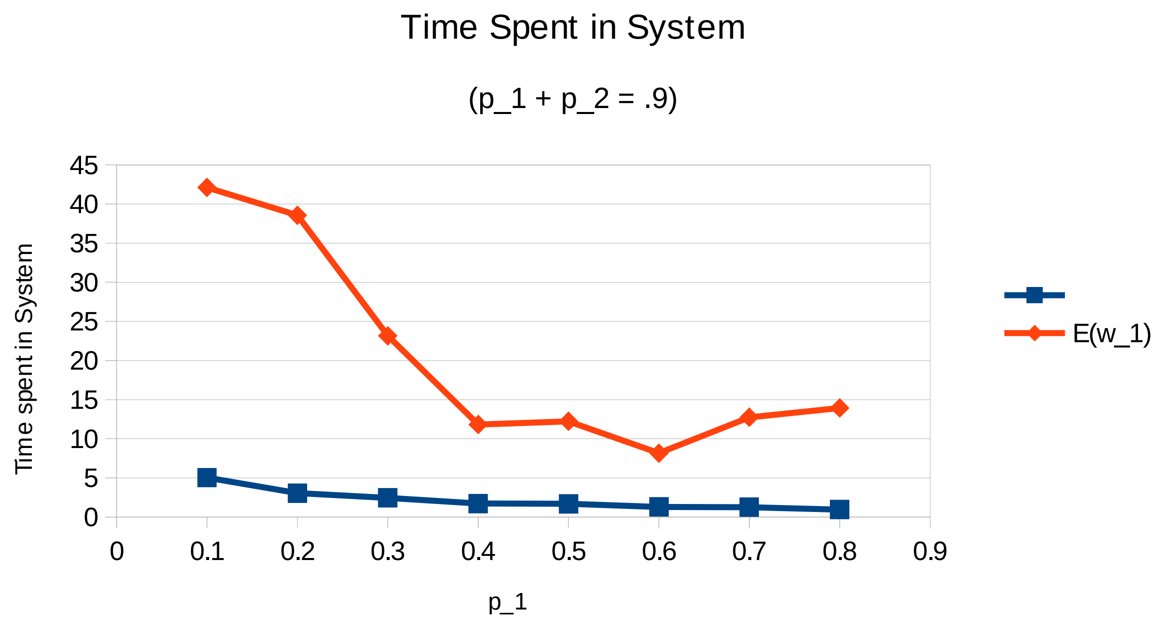
E(N_1)	E(W_1)	E(N_2)	E(W_2)
0.253784	1.20722	2.531767	4.71796
0.267171	1.19377	2.089839	4.42790
0.281736	1.30822	2.465020	4.71225
0.242819	1.17138	1.832631	3.78892
0.266560	1.34248	2.179017	4.39502

Averaged Simulation Results:

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



Sorry, I couldn't get the graph's legend to be correct. Blue is E(n_1), Orange is E(n_2)



Sorry, I couldn't get the graph's legend to be correct. Blue is E(w_1), Orange is E(w_2)

David Perry CS 543 HW#2 October 19. 2014

3.)

This problem required the implementation of the card piling problem using a genetic algorithm. The card piling algorithm I solved assumes that there is a deck of 10 cards. Each card in the deck has a value from 1 to 10. To solve the problem the program must find a way to separate the cards in one of two piles. In one pile the sum of all the values of the cards must equal 36. In the second pile of cards the product of all the card values must equal 360.

The process of solving the card piling problem through use of a genetic algorithm is as follows:

- Create a representation for a potential solution. Essentially, a bit array is created representing which pile a card is placed in.
- Multiple representations are created randomly. This pool of potential solutions is called population.
- The population competes against each other to see how has the best or closest solution. This is computed through the use of a fitness function. Essentially, the program calculates how large an error the solution has.
- The winners of this competition are "mated" to create new potential solutions.
- With some small probability members of the population are mutated. To accomplish this a random bit is flipped. This provides a way for the algorithm to escape a local maximum.
- This whole process is completed repeatedly until a acceptable solution is found.

My program is implemented in C but was modeled after a C# solution found at the link:

http://www.codeproject.com/Articles/16286/AI-Simple-Genetic-Algorithm-GA-to-solve-a-card-pro

An example solution generated by my program is:

```
After 743 tournaments, Solution sum pile (should be 36) cards are :
2
7
8
9
10

And product pile (should be 360) cards are :
1
3
4
5
6
```