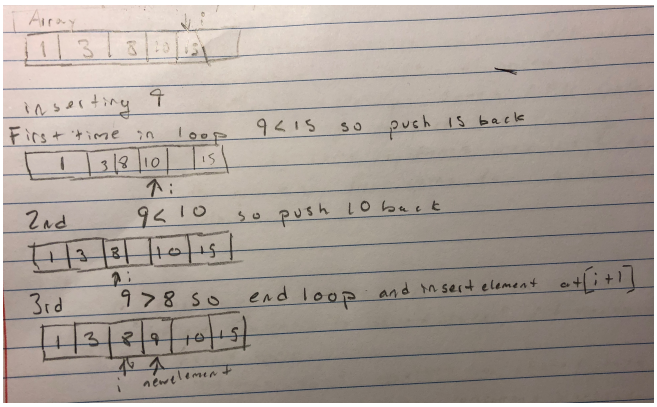
The first step I took to creating the Discrete Event Simulator was creating the queues:

I did this because once I have the queues made all I would have to do was glue everything together. Each queue is made up of a structure with three elements. The elements per structure are the name of the job, the time of the job, and the id, where the id represents where the event will go next.

The priority queue was the hardest to make which is why I went online to find a priority queue API. I found one the made some changes to it so I could input structures into a priority queue. My priority queue holds every element in it inside of an array. It has two methods, PQinsert() (inserts a new element into the priority queue) and PQremove() (removes the element with the highest priority).

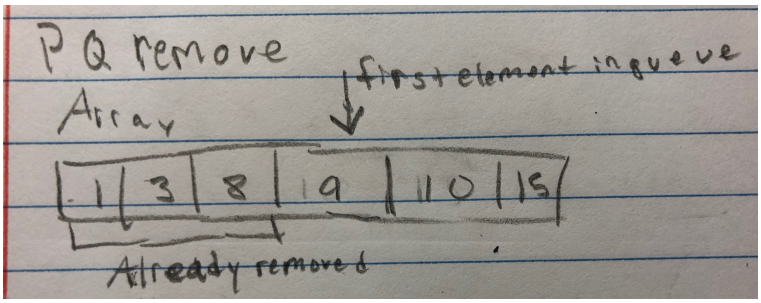
PQinsert() takes in three parameters, the name of the job, the time of the job, and the id of the job. To find the location of the element, I start at the back of the array and move to the front. First I set a variable i equal to the number of elements which I already inserted into the array. Next, I run a loop that will run while the current element in the arrays time is higher than the time of the element I am inserting and while we are not at the front of the array. In the loop, I push everything in the array back one as well as subtracting i by one because I am moving though the array backward. Once I break out of the loop, i+1 will be pointing to the position in the array that the new element will be inserted into. Therefore, I will insert all elements the name, time, and id of the element in the i+1 location of the array.

A diagram of how it works is below:



To remove the element with the highest priority, I call the method PQremove(). So every time I call the remove function I increment the number of elements removed and return the first element in the queue. I don't remove it from the array, I just keep track of the number of elements I remove, and that number will point to the first element in the array. I decided to do it this way to avoid complications of actually removing an element, and I also found this method to be more effective and simpler.

A diagram of how it works is below:



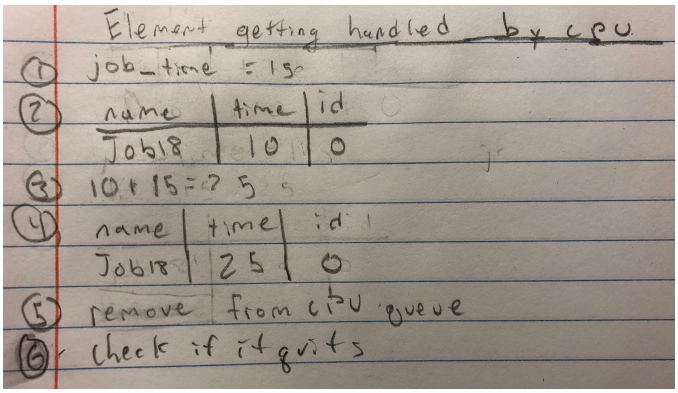
The next step I took was creating the first in first out queues needed for the CPU and the disks. I created a structure which holds the elements of the job and created an array of structures for the CPU and the disks. The CPU and the disks all have 3 methods for each of them, an insert method, a remove method, and a handler method. All of these methods work the same way just with minor tweaks for each server.

The insert method for each server is the same. For the CPU insert method, it runs a while loop until it reaches the end of the array. Inside the loop, it increments i by one. Once we leave the loop, it will insert every element of the job into location i of the array. Lastly, it gets sent back to the priority queue with an id pointing to its location and where it will go next. It will then wait until its priority is the highest so the disk can process it. The insert method for the disks works the same way.

The remove method for the servers works the same way the remove method worked for the priority queue.

The handler method for the CPU will take in three parameters, job_time, quit_prob, and quit. Where quit is a random number I generate from 0 to 100 and quit_prob is the probability that the job will quit once it leaves the CPU. First, the job_time gets added to the first element in the queue's time; then the remove method gets called to remove the first element in the queue so the CPU can process it. Then I change its id to 1 to show that it has been removed from the queue and will be processed by the CPU. Lastly, I check if the element will quit. If it quits i store the element in the first position of the array so it will not be used again.

A diagram of how it works is below:



The handler event for the disks only takes in job time as a parameter. Also, since a job cannot quit after it leaves the disks that part is not in the handler for the disks. Lastly, at the end of the handler method for the disks, I insert it back in the priority queue with an id of 0 so it will be put back in the CPU's queue.

Once I created my queues all I had to do was put everything together like this:

My main program starts off by initializing the global variables that are read in from a text file using fscanf. After initializing all variables, I insert two elements into the priority queue. The two elements inserted are Job1 which is initialized to INIT_TIME and simulation finished which is initialized to FIN_TIME. After that, I run a while loop that will end when the current time is greater than or equal to the simulation finished time. Inside the while loop, I have a switch statement for deciding if I have to create a new job or process an event. Case 1 created a new job and inserts it into the Priority queue and case 2 processes the jobs. Once case 2 finishes, it set option equal to 1 so the next time around we will go into case 1 and create a new event.

These two switch off between processing events and creating a new job because after we process an event the next step is creating a new job.

Case 2 is the bulk of the program and is how I process the event. The first thing that happens in case two is removing an element from the priority queue. Then there is another switch statement that will switch to a case based off of the id of the element which was just removed.

Case 0 places an event we removed in the CPU queue and increments the number of elements in the queue by 1.

Case 1 processes the event which is first in the CPU queue. First, it generates a random job time in between the min and max and generates a random quit number in between 0 and 100 to see if it will quit after it leaves the CPU. Next, it runs the handler method for the CPU to process the event. After that, the job time will be added to the current time, the CPU time and the total events time and writes the data to the log file. Lastly, the event will be placed into one of the disks. This is done by checking which disk currently has the least elements and placing the event in that disks queue. I insert it into the queue the same way I explained above for case 0.

Case 2 and case 3 are the cases for processing events in the discs. First, I generate a random job time and add that to the current time, disk time, and event time. Next, the handler method is called for the disk to process the event. Lastly, I write the data to the log file.

Once we are done processing the event, I change the option to 1 to create a new event the next time we go through the loop. Next, I have 3 if statements to find the max number.

Once time is greater than or equal to the finish time we exit the while loop and print the stats at the bottom of the log page. Then we close the log file and the simulation is over