NOT DONE

DONE BUT NOT TESTED

DONE AND TESTED

Random Number Generation

Random number within a range

Random number between 0 and 1

Random number for Work of Type II:

* Range [200, 600], mean 350
* Generate a value *num* in [0, 1]
* Use *value = ceiling[200 + 400\*num^(5/3)]*

Random number for Work of Type III

* Range [150, 500], mean 250
* Generate a value *num* in [0, 1]
* Use *value = ceiling [150 + 350\*num^(5/2)]*

Random number for interval for Type III processes

* Range [25, 75], mean 40
* Generate a value *num* in [0,1]
* Use *value = ceiling [25 + 50\*num^(7/3)]*

Random number for length for Resource C

* Range [100, 400], mean 200
* Generate a value *num* in [0,1]
* Use *value = ceiling [100 + 300\*num^2]*

Creating Processes

1. Generate a priority
   1. 1 - 50%
   2. 2 - 20%
   3. 3 - 10%
   4. 4 - 10%
   5. 5 - 5%
   6. 6 - 5% 🡨 highest priority
2. Generate random number 1 – 100
   1. 1 – 50 🡪 Type I
   2. 51 – 80 🡪 Type II
   3. 81 – 97 🡪 Type III
   4. 98 – 100 🡪 Type IV
3. Determine work required
   1. Type I 🡪 generate a random number between 0 and (75 – 25) add to 25
   2. Type II 🡪 use the random number generator for the work of type II
   3. Type III 🡪 use the random number generator for work of type III
   4. Type IV 🡪 generate random num between 0 and (1000-400) add to 400
4. Determine blocking
   1. Type I
      1. Will it block 🡪 generate random number between 0 and 1, <0.4 block
      2. Where it blocks 🡪 pick number between start + 1 and end -1
      3. Resource used 🡪 random number between 0-1, <0.5, A, otherwise B
      4. Length of block 🡪 resource dependent
   2. Type II doesn’t block
   3. Type III
      1. Blocking interval
         1. Non-uniform distribution, use random number generator for this type to determine when to block, length described below
      2. Resource used 🡪 random num between 0-1,
         1. x < 0.5 🡪 B
         2. 0.5< x <0.85 🡪 A
         3. x >= 0.85 🡪 C
      3. Block length 🡪 resource dependent
   4. Type IV
      1. Blocking interval 🡪 uniform between 40 and 120
      2. Resource Used 🡪 B 50%, C 50%
      3. Block Length 🡪 resource dependent
5. Blocking length
   1. A 🡪 uniform between 60 – 100
   2. B 🡪 uniform between 75 – 125
   3. C 🡪 non-uniform uses its own generator
   4. D 🡪 irrelevant
6. Report Function
   1. ID
   2. TYPE
   3. ARRIVAL TIME
   4. RUNNING TIME
   5. LIST OF BLOCKS
   6. TOTAL BLOCKING TIME

Report for entire Process Set

* + - 1. List number of processes
         1. Sublist for each process type
         2. List all process reports
      2. Total Run Time
      3. Resources
         1. Total number of blocks
         2. Total blocking time
      4. Total Block Time
      5. Average Interarrival time (how often a process arrives on the cue)

THE SCHEDULER

Queue

All queues need a way to get the next thing, and also to remove the next thing because you’re taking it off the queue (essentially just pop)

They also each need a method to say if they are empty, and there should also be some way to add to the queues, and to sort those queues (where the plain queue would not require any sorting)

1. Priority Queue
   1. Every priority queue should have its own sort method. There should be no problem sorting the method so long as you can use a different function as the predicate (because the way that we sort them predominantly has to do with length etc)
2. Plain Queue
   1. Does not need any sorting function, and to add something in really only requires one thing, the arrayList add

Schedulers