Output from main(): N/A

1. I designed a system that uses three classes and a function to simulate the process scheduling of given programs. The classes were a Linked List and a class of Nodes for the implementation as well as a Process Scheduler class. The input was read in from a file and fed into the process scheduler, which internally implements a linked list. The run\_processes method is then called, and the output is fed to a .txt file of a generic name to be relabeled later. For the policy of Shortest Policy Next, a quicksort sorting method was used on the linked list and then the entries popped off. This ended up being quite slow for large data sets as the quicksort algorithm incorporates quite a lot of lookup, the time of which for a linked list is, at worst case, O(n). This was substituted with a binary tree sorting method, implementing a Binary Tree class, a Binary Tree Node class and a function to implement the two for sorting. This turned out to be a huge improvement as it requires little to no lookup.



1. As shown in each of the input sets, and as one would expect, the policy that results in the most processes being handled in the shortest amount of time (aka the lowest mean time to completion) is the Shortest Process Next policy. However, this process also has a very high standard deviation, as the midpoint is set low, so many processes are finished early on but then the longer, slower ones are next in line and take longer. This can be useful for doing as many tasks as possible in a short amount of time, assuming all are equal priority. Note, however, that if processes are added in the middle of processing, it is possible for the longer processes to never be reached. This is the main strength of the First In, First Out policy, which has similar but inferior stats to the SPN policy. The last policy, Round Robin, looks much worse on paper as no process gets to progress further than the quantum limit before the rest of the processes are sent through again. This, however, is the policy with the lowest standard deviation, which can be useful as the time at which the processes finish execution is all relatively close to one another and not as exponentially spread out as the Shortest Policy Next.
2. When the quantum limitation exceeds any required time for any process in the Round Robin policy of the Process Scheduler, the policy becomes identical to FIFO with each process being handled in order of arrival.
3. Sample outputs (randomly selected) from an input set put through the process scheduler with policies FIFO and RR(200) (the largest quantum is 110):

FIFO:

…

Data: A164 (tS: 0) Lim: 200 Time: 10134.

Data: A165 (tS: 0) Lim: 200 Time: 10156.

Data: A166 (tS: 0) Lim: 200 Time: 10255.

Data: A167 (tS: 0) Lim: 200 Time: 10270.

Data: A168 (tS: 0) Lim: 200 Time: 10366.

Data: A169 (tS: 0) Lim: 200 Time: 10475.

Data: A170 (tS: 0) Lim: 200 Time: 10508.

Data: A171 (tS: 0) Lim: 200 Time: 10580.

Data: A172 (tS: 0) Lim: 200 Time: 10674.

Data: A173 (tS: 0) Lim: 200 Time: 10740.

Data: A174 (tS: 0) Lim: 200 Time: 10847.

…

RR(200):

…

Data: A164 (tS: 70) Lim: 0 Time: 10134.

Data: A165 (tS: 22) Lim: 0 Time: 10156.

Data: A166 (tS: 99) Lim: 0 Time: 10255.

Data: A167 (tS: 15) Lim: 0 Time: 10270.

Data: A168 (tS: 96) Lim: 0 Time: 10366.

Data: A169 (tS: 109) Lim: 0 Time: 10475.

Data: A170 (tS: 33) Lim: 0 Time: 10508.

Data: A171 (tS: 72) Lim: 0 Time: 10580.

Data: A172 (tS: 94) Lim: 0 Time: 10674.

Data: A173 (tS: 66) Lim: 0 Time: 10740.

Data: A174 (tS: 107) Lim: 0 Time: 10847.

…

This shows no difference between a limit of 200 quantums and no limit (aka a limit of 0). (The time marker is that of the internal clock, with the tS denoting the number of time slices of that process.)

The output from the process scheduler:



As clearly seen in the above table, the values for the two outputs are identical.