**Operating System Simulator Re-Design**

Student Number: 28351 Grader Number: 96260

**Design Type**

The design used was a high level design. This means that it strictly supports a round robin scheduling scheme.

**Key Features and Structures**

The main data structures used to store information for scheduling purposes were linked-queues. This minimalistic choice of data structure design allowed for simple implementation and reduced the need for limiting the numbers of objects the simulator could handle, increasing the degree of multi-programming.

The design uses a pair of queues – one to manage CPU bound processes, and one to order all I/O bound operations, and all interrupts are handled before a running process is given a CPU cycle.

Should any other data need to be held, it is held in precisely sized arrays.

Processor threads are used to simulate the raising of an I/O completion interrupt.

Timing is done using “wall-clock” time as opposed to the CPU clock of the processor running the simulator.

**Overall Flow of Simulator**

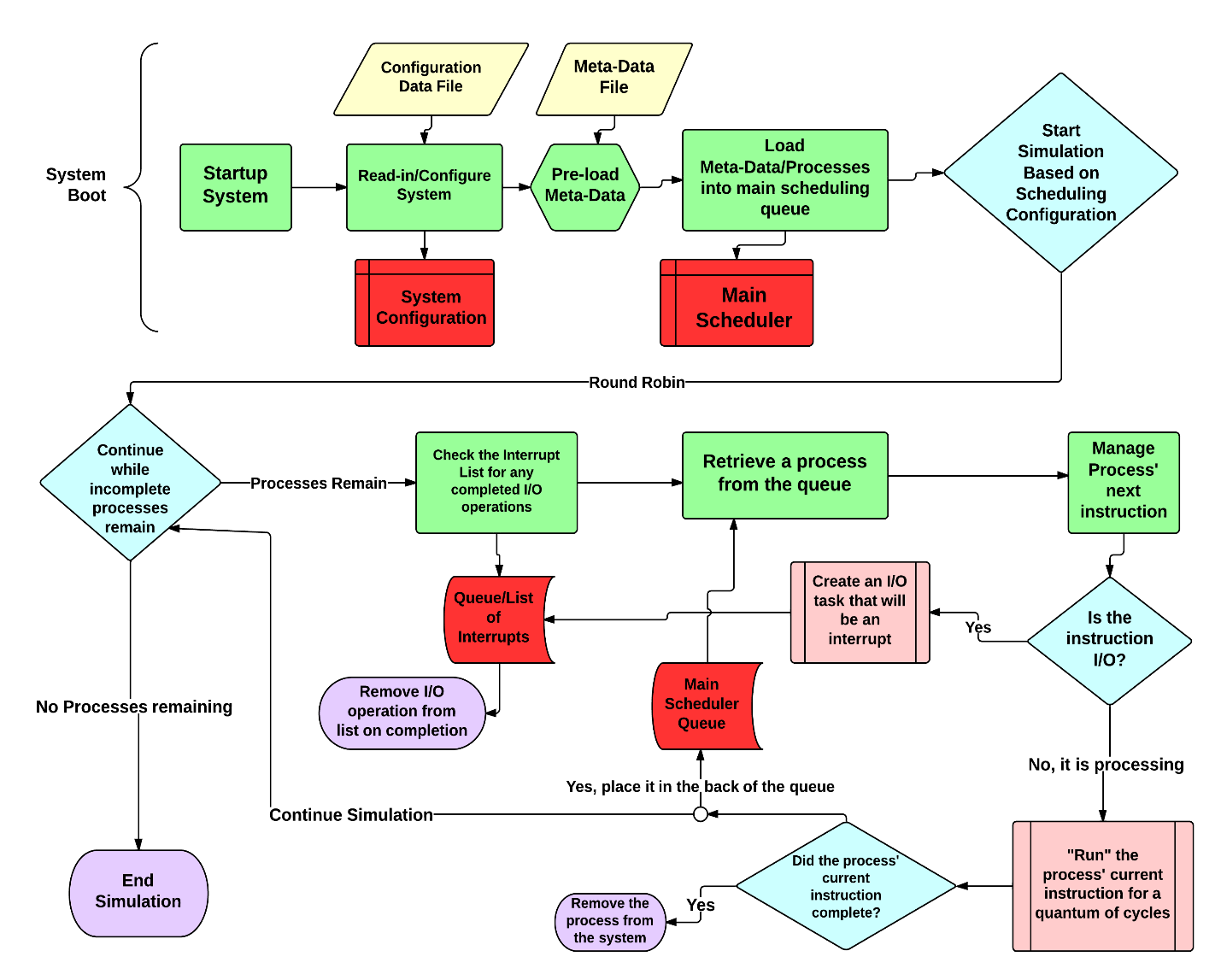
The simulator starts by reading in all data. Configuration data is read into the program in the expected manner. Then the meta-data is read into an array as a sort of preparation operation, and then it is processed in a way that splits the meta-data into instruction sets for a process structure.

Once the meta-data file is loaded, simulator execution begins. This design was made to use a different simulation mode entirely, depending on the schedule scheme specified in the system configuration. Since round robin scheduling is all that was required, this was the only method implemented. Otherwise, an error would be reported and the simulation would end.

During the simulation, all I/O interrupts are checked for at once, and handled if an interrupt is detected. Interrupts are checked for at the beginning of a time quantum, not at the beginning of a CPU cycle, resulting in a low priority interrupt system where running processes are not allowed to be actually interrupted.

When processes reach the front of the queue and are pulled for processing, and handled by a software method that examines the next instruction in the process and depending on the process’ next instruction, it is decided at that time what will happen with that process. If the instruction is an I/O one, an I/O procedure is started, otherwise the process is given CPU time.

Processes with a processing instruction execute that current instruction one quantum at a time, being re-queued after each quantum, keeping with the definition of round robin scheduling. This does, however, imply a “low” priority for interrupts, since they can only arise after a quantum, not after every cycle.



**Other Remarks**

Some of the components seemed to be less modular than might be desired, but given the scope of the assignment (or perhaps the implementer’s desire to reduce computational overhead), it is understandable why some of the data structures’ functionality was coupled to the functions that made use of said data structure. That being said, this was not in the spirit of the assignment, the spirit of an easily expanded design/implementation.

The design could have also been simplified if the (basically) raw meta-data were not used for storing instruction sets, but rather some sort of symbolic representation within the program.