Michael Battaglia

CS526

Final Project Documentation

**Description of Data Structures:**

* The main data structure of this program is the *processPriorityQueue* priority queue that utilizes a comparator to compare the priorities of processes. As “time” advances in the program, processes are brought from a holding area (the *processData* array list described next) into the queue based on the current program time and arrival time of each process. A process will not enter the queue until its arrival time has been reached. In order to prevent process starvation, a maximum wait time is implemented that will decrement a process’s priority by 1 if its wait time exceeds the maximum wait time (default 30).
* *processData* is an array list of processes that stores information about a process that is needed to execute that process in *processPriorityQueue*. A process is a user-defined class that holds a process’s: ID, priority, arrival time, duration (how long the process takes to “execute”), waiting time, and overall total waiting time so far. When a process’s arrival time is reached in the program, it is moved from *processData* into *processPriorityQueue* and either executed immediately (if there are no other processes currently in the queue) or forced to wait until both the currently running process finishes and no other processes with a lower priority arrive in the queue.
* The final data structure used in this program is the array list *waitTimeAvg*. This structure’s sole purpose to keep a record of the wait time of each process in the queue. At the conclusion of the program (when both *processData* and *processPriorityQueue* are empty), the individual wait times of each process stored in *waitTimeAvg* are averaged out to find the total average wait time of each process.

**Observations & Learnings:**

The main thing I learned from this project is to code in “chunks” and to not try and write an entire program in one sitting. In the past, I have been able to get an assignment mostly working in a few hours, but I knew that for this project I would have gotten too overwhelmed when I encountered a bug so I forced myself to code small pieces of functionality and test them thoroughly to make sure nothing broke when I added something new.

An interesting observation I had when manipulating the maximum wait time was that the total wait time, average wait time, and order of process execution remained unchanged when the max wait time was between 1 and 79. The only thing affected by changing the max wait time between these bounds was the amount of times a process’s priority got changed. Decreasing the maximum wait time to a low value did clearly show how Java arbitrarily decides which process in the queue goes next if two processes have the same priority. I would have assumed that Java would take the process duration into account when choosing between two processes with the same priority, but that was not the case which shows that Java’s choice is truly arbitrary. This occurred when the max wait time was 30 as well, but I felt it was more apparent with a lower max wait time.

When I increased the maximum wait time to 80 or higher, the order of processes being executed did change and process 10 was no longer the last process to be removed from the queue (process 5 was). A high maximum wait time really shows how a process can be “starved” if processes with lower priorities keep arriving; process 5 arrives at time 40, but was not executed until time 170 when the max wait time is over 80.

Finally, the fact that certain components of the output were not affected by manipulating the maximum wait time suggests that there is a “range” of acceptable max wait times when designing a program that implements a process queue. I know that this was a simple implementation of process execution, but I would imagine that in a real implementation of a program similar to this, more variables would have to be considered in addition to the maximum wait time in order to optimize the performance of the queue.