**CPU Scheduler by Daniel Rossano**

This program serves as a way to simulate two different CPU scheduling algorithms: First Come First Serve (FCFS) and Shortest Job First (SJF). To make the program easy to use, the following assumptions have been made:

* All processes arrive at the ready queue at the same time
* We know the burst time for each process
* All processes have a burst time between 1 and 1000 milliseconds
* No new processes can enter the ready queue *while* other processes are being scheduled

The user’s interaction with the program is handled entirely by the CPUInterface JForm. Any algorithms used to emulate a CPU are done within the CPUAssignment class.

**The CPUInterface JForm**

The CPUInterface contains makes frequent use of the following two variables: a CPUAssignment, *processDeterminator*, which is used to invoke the various methods of CPUAssignment from events triggered by the buttons in CPUInterface, and a 2-dimensional object array, *tableFace*, whose contents are output to the JTable processOrdering. When a new process is created, an object array p of length 4 is created. Next, *processDeterminator* invokes either the customProcess or randomProcess methods in the CPUAssignment class, depending on which type of process the user is trying to create. Once these methods have returned an object array (see the CPUAssignment section for the contents of the array) CPUInterface expands the array *tableFace*, adds whichever process was returned, and outputs the contents of *tableFace* to the JTable. If any algorithms have been run prior to adding a process, the outputAverages text will change to inform the user that the previous averages are no longer accurate.

When one or more processes are removed, *processDeterminator* will invoke the removeProcess method, scale the array tableFace down by one index, and output the remaining processes stored in tableFace to the JTable. If a scheduling algorithm has run prior to the removal of a process, the JTextArea outputAverages, will notify the user that the average wait and turnaround times are no longer accurate. Also, if SJF has run prior to the removal, the processes will be sorted from first to last, and then the last process will be removed.

Lastly, when the user wants to run a scheduling algorithm, the JComboBox algorithmSelection is used to determine whether the user wants the processes to be scheduled according to FCFS or SJF. Because both of these algorithms return a 2-dimensional object array, the two CPUAssignment methods getAvgWait and getAvgTurn must be invoked so that the program can output the average wait time and average turnaround time to the JTextArea outputAverages.

**The CPUAssignment Class**

Anything not seen by the user is done in the CPUAssignment class. The randomProcess and customProcess methods will take an object array *p*, which will store all of the necessary information for a process. First, the process is “named” by simply concatenating the string “P” with the current value of the instance variable processNumber, which of course tracks the number of processes currently available. The next item in the array *p* is the burst time for the process. The burst time will always be between 1 and 1000, but in the randomProcess method, the burst time is generated randomly, while in the customProcess method, the user is allowed to input the burst time manually through a dialog box. The object array *p* is then given a wait time and turnaround time, which are always set to “N/A” by default, since no algorithms have run during its creation. Lastly, the object array *p* is returned. The array *p* may return null only in the customProcess method, and only if the user has entered an unacceptable (noninteger or not between 1 and 1000) value.

In the firstComeFirstServe method, a 2-dimensional object array is entered. This program was intentionally designed so that the contents of the array would contain the processes from first to last. A simple for loop will be enough to establish each process’ turnaround time and wait time by reading them from the array that was input. These values are stored in the instance variables *turnaround* and *waitTime* respectively. The averages of these values are then computed and stored in the instance variables *avgWait* and *avgTurn*. If the SJF algorithm has run prior to FCFS, the processes will be reordered from first to last before FCFS is computed. This is because the SJF algorithm will reorder the process by burst time, and so the processes may not be in order of first to last. Once the average wait and turnaround times have been computed, the 2-dimensional object array the was sent in gets returned.

In the shortestJobFirst method, a 2-dimensional object array is entered. Since arrival times are assumed to be uniform, we only need to sort the processes by burst time. Using a bubble sort, each process in the given array is swapped based on burst time. The rest of the algorithm is identical to FCFS, as the processes are now sorted from shortest to longest burst time. Additionally, after the bubble sort, an instance variable *sorted* is set to true so that if FCFS is run, or a process is to be removed by the user, the processes will be reordered from first to last. To prevent this reordering from occurring each time FCFS is run or each time a process is removed, the Boolean is set to false after the processes have been reordered.

Lastly, the getAvgWait and getAvgTurn methods are used by CPUInterface to output the average wait and turnaround times to the outputAverages textbox. These methods are invoked immediately after SJF or FCFS has been run, and cannot be accessed by the user. They are simply get methods for *avgWait* and *avgTurn*. After these values have been stored, they are set to 0 so they can be reused next time a scheduling algorithm is run.