# ICSI412 Operating Systems – Project 3 – Prioritize the Scheduler

“All animals are equal, but some animals are more equal than others.”

Animal Farm – George Orwell

## Overview

Some processes need to be scheduled as soon as possible. One example might be a process that is playing music or a movie. Some processes need to be scheduled quickly, but not absolutely top priority. An example might be your web browser. Finally, some processes can happily run in the background whenever they can get some time. An example of these is getting weather updates.

You might naively start out by saying that when we need a process to run, we will do anything on the realtime list, then anything on the interactive list, then anything on the background list. The problem with that is that some processes will never get run time. This can lead to deadlocks and unresponsive user interfaces. We will be implementing a probabilistic model. If there are realtime process, then 6/10 we will run a realtime process, 3/10 we will run an interactive process and 1/10 we will run a background process. Otherwise, if there are interactive process(es), we will ¾ run interactive and ¼ run background.

You might notice that a user can abuse this system – just mark **their** process as “realtime”. Then they get more runtime than others. We are going to fix that by watching realtime and interactive processes. If they run to timeout more than 5 times in a row, we will downgrade their priority (realtime 🡪 interactive, interactive 🡪 background). That downgrade is permanent for the life of the process, although I am sure you can see how we could “promote” them back up later.

## Supplied Classes

**OSInterface** has changed the signature of CreateProcess.

**PriorityEnum** contains the different priorities.

**Task 1** – change OS to accept an additional parameter for priority

Since each process needs a priority, we will need to supply that when we create a process.

**Task 2** – Create PriorityScheduler

This new class will replace BasicScheduler. Feel free to copy/paste some functionality from BasicScheduler. This class will have multiple work queues (again, not necessarily Java queue), one for waiting and one for each priority.

**Task 3** – Update CreateProcess and DeleteProcess

CreateProcess now needs to take into account priority and put the process in the appropriate queue. One thing that you will need to deal with – when you take a process out of the queue and run it, you will need to remember what queue it came from. Consider a solution similar to the solution for remaining milliseconds to wait. DeleteProcess now needs to do more work to find a process to delete.

**Task 4** – Picking a new process to run

As described above, use a Random class to chose a random number. Choose the appropriate queue to take the next process from. Make sure that you check to make sure that there is actually something in that queue. Ensure that after a process is done running you put it back in the appropriate queue.

**Task 5** – check for downgrades

If a process is elevated priority but is “abusing” the system (as described above) downgrade its priority appropriately.

**Test your code!**

***You must submit buildable .java files for credit.***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rubric | Poor | OK | Good | Great |
| Update OS | No Changes (0) |  |  | Now accepts priority (5) |
| PriorityScheduler | No Changes (0) |  |  | Has appropriate queues (5) |
| CreateProcess | No Changes (0) |  |  | Inserts into appropriate queue (20) |
| DeleteProcess | No Changes (0) |  |  | Deletes from all queues (10) |
| Choosing new process to run | Not probabilistic (0) | Checks for queue emptiness (5) |  | Probability assignment is correct (20) |
| Priority Downgrades | None (0) |  |  | Correctly downgrades priorities (20) |
| Post-Wait queues correct | Non correct(0) |  |  | Correct (20) |