CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. There are many different CPU-scheduling algorithms, for example the **priority-scheduling** algorithm is based on the priorities associated with each process. The CPU is then allocated to the process with the highest priority. In the case of equal priority, processes are scheduled in first come first serve order.

Note that we discuss scheduling in terms of ***high*** priority and ***low*** priority. Priorities are generally indicated by some fixed range of numbers, such as 0 to N. However, there is no general agreement on whether 0 is the highest or lowest priority. In our project, 0 is regarded as the highest priority.

Priorities can be defined either internally or externally. Internally defined priorities use some measurable quantity or quantities to compute the priority of a process: time limits, memory requirements, etc. External priorities are set by criteria outside the operating system, such as the importance of the process as is set in our programming assignment.

Priority scheduling can be either preemptive or nonpreemptive. When a process arrives at the ready queue, its priority is compared with the priority of the currently running process. A preemptive priority scheduling algorithm will notify the CPU if the priority of the newly arrived process is higher than the priority of the currently running process. A non-preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue. In the case of our programming assignment, we are making a preemptive scheduling program that will compare priorities the newly arrived thread with the currently running thread.

A major problem with priority scheduling algorithms is **indefinite blocking**, or **starvation**. A process that is ready to run but waiting for the CPU can be considered blocked. A priority scheduling algorithm can leave some low-priority processes waiting indefinitely. The solution that we will be using to tackle this problem is by incrementing the priority number of any process that has been granted two time slots (higher priority number = lower priority).

Code Structure:

Queue wrapper class