OS CW Task 1: Advanced

In the multi queue scheduling protocol, if higher priority queues are full then the queues beneath it are starved for any CPU resources. This could lead to a poor user experience as the OS will feel less responsive to “normal” actions if there are lots of higher priority threads queued. So we want a way for these lower priority threads to be given resources while still maintaining the priority of of the queues. To do this I did some research and found out about a scheduling algorithm known as lottery scheduling (https://en.wikipedia.org/wiki/Lottery_scheduling) where each process is given a number of lottery tickets depending on its priority, and higher priority processes are given more lottery tickets than lower priority processes. Then the scheduler will pick a random lottery ticket from the total pool of all process lottery tickets and the owner of that ticket will be assigned CPU resources. So in theory the higher priority processes will be chosen more than the lower priorty ones and thus have control of the CPU for more time. Unfortunately in InfOS there is no functions for generating random numbers so I couldn’t proceed with this idea. Instead I implemented a weighted round robin system where the higher priority processes hold onto control of the CPU for longer than the lower priority processes. This algorithm has a similar aim to lottery scheduling but instead of using randomness just gives fixed slices of time to each priority queue. This means that the lower priority processes do not get starved for CPU resources but also that the higher priority processes should be given more access to the CPU. This algorithm has a similar aim to lottery scheduling but instead of using randomness just gives fixed slices of time to each priority queue. Currently, my algorithm gives 8 “ticks” to the REALTIME queue, 4 to the INTERACTIVE queue, 2 for NORMAL, and 1 for DAEMON although this could and should be edited in the future to balance out how much you want to give CPU time to the lower priority processes.

Drawbacks:

-The processes within the multiple priory queues are still scheduled in a round robin fashion which means valuable time is lost performing context switches on every tick.

-If bad time slices are chosen this could starve the important high priority processes being scheduled.