Program Assignment 2

Ethan Lamitie

CSI-385 Operating System Architecture
Ahmed Hamed, Ph.D.

Introduction

The purpose of this assignment was to simulate a process scheduler by learning the different algorithms that the scheduler uses in real world operating systems. I think it was a very successful exercise in learning about how the scheduler operates, and unrelated to the assignment topic, I think it helped me with my ability to develop algorithms from just concepts, specifically the Round Robin algorithms.

Data Implementation

I chose to use a simple struct to define what a 'process' is in my program. In my program, a process has an id, execution time, priority, and a flag as to whether the process has completed execution or not. The ids are unique, incrementing up from 0. The execution time is a random number between 1 to 15 [inclusive], and the priority is also a random number, but between 1 to 3 [inclusive].

I also designed an extremely simple wrapper for a standard array, to make it easier for me to pass it around to various functions that would operate on the processes. It's nothing more than a struct called 'PArray', and it has an array of 10 process pointers.

Algorithm Details

Starting with FIFO, it's dead simple. It initializes a single process, and then in a for loop "forks" 9 other processes off of the first one. They obviously aren't real forks, but rather just part of the simulation. Once the array is set up with that data the next function call executes the array.

Shortest Time and Priority are virtually the same algorithm working on different data. Both are basically a Bubble Sort that sorts the processes in increasing order based on either their execution times, or their priority numbers.

The Round Robin algorithm doesn't particularly care about how the processes are sorted, it just executes them in Round Robin fashion in the order they were given. As such, the array needs to be sorted *before* handing it off to the Round Robin algorithm. What it does is essentially have a while loop that checks if there are currently processes waiting for their turn to be executed in the scheduler, and if there are, it goes through every process, executes each process for the time given (5 per process in this implementation), and will set the process flag 'completed' to true if the process has a remaining time of 0. Before doing any of this it checks if a process has been completed so it's not operating on processes that it should be operating on. If there are no more processes waiting to be executed, the algorithm is done. It checks if there are processes waiting by calling 'hasWaitingProcesses()' on the array which just loops through, and if it finds a process who's completed flag is false it returns true.

Results

Due to this being simply a simulation, it's hard to necessarily extrapolate data worth analyzing about actual process execution times, turnaround times, and delays, but it is worth noting that research and learning about how the algorithms work has lead me to the conclusion that sorting things around Round Robin execution is the best solution, although in my eyes it doesn't necessarily matter to me how the processes are sorted before that execution. It's a nicely organized way of handling each process, while giving every process the equal chance to the resources that it needs from the operating system.