Overview

For this approach, I attempted to implement a solution in python using QtPy5. Using PyInstaller, I also tried to make it a single executable that you would have to simply click and run, but that was leading to some issues with actually getting it to work. It is still included, but I honestly do not know if it will work or not for you. If it doesn't work, you can still easily demo and look at the code if you just open it in a python editor and running main.py. I have also included a readme text file that should have all the modules I used in the environment.

Classes

Main – The heart of the program. Connects to the QtPy ui file (mainwindow.ui) and generates the window. Hitting the start button simulates the running of the desired amount of processes. Unfortunately, when looking at the GUI you will see a lot of extra features I tried to get working. Currently nothing displays correctly but the code does indeed run, and values do change. The critical section resolution section is here in the form of isCritical, which is a tuple that holds a Boolean value and and pid. When isCritical is set to (True, <pid>), no other programs are allowed to start running until the it goes back to its non-critical value of (False, -1). It gets set to critical while a process is running, and it gets to the instruction marked critical in its code.

It simulates running a program by decrementing 1 from the current processes instruction cycle time, if the time hits 0, it goes through a multitude of checks to determine what it should do next.

Process – builds process file, along with all of its attributes. Randomly decides a critical section by just picking random instructions to be the start and end point of the section.

Scheduler – A simple round robin scheduler that takes the average of each processes burst times, adds them together, and then averages that again to make up the quantum time for a program. Using that algorithm, sometimes the quantum will end up being zero, so to remedy this I added a failsafe that if the quantum was below 5, set it to 12 to ensure the simulation will still run

PCB – Holds the pid (random integer between 0 and 10000), the type, and the current instruction position of the process, and if its currently in critical state

Enums – Holds the values for the different states the processes can be in, and the different types of instructions the templates can hold