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 number 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,), 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.

Removed the use of isCritical and replaced it with native python acquire and release. Couldn't quite figure out how to get programs to terminate completely so I included a break statement at the end that stops the program to prevent looping forever and you can at least see things are happening.

Changed main running method to make the initial set of processes (10 now), add either at the most 4 processes to the running queue (or however much will fit in the memory), the processes all run on separate threads, the main thread waits for these processes to complete, once they complete, the state of the processes is determined, and then either new processes are added from the new queue again, or if the new queue is empty, the processes in the ready queue are added to the running queue and the cycle repeats

Methods –

updateClicked

Changed from constantly forever looping thread to thread that just goes through all the processes once, also makes the main thread wait for it to finish before it keeps going because it would overflow when accessing the list while the processes were being changed.

threadRun – Basically the old run method from phase 1, but this time I redid the logic a bit that should account for all scenarios, added a 5% chance for a IO interrupt to happen, which would make the thread sleep for 5 seconds (I originally had it at 15% and 7 seconds but that got very annoying when trying to debug). Added a branch for if the next instruction is a fork, create a child process (exactly the same as main process with pid incremented by 1 and its on the next instruction (the instruction after the fork))

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

Added a small bit of code that makes a memory value that is the result of the number of instructions multiplied by .5 and rounded down. This combined with the fact the number of instructions is random should lead to multiple memory values being made from any template

---

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

Added a list object that holds all the children pids for cascading termination

---

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

No changes this phase

The link to my github is https://github.com/Dwr3k/312-Project and there should only be one commit there for you to download