Project 2: Preemptive CPU Scheduling

Due date: Midnight of Wednesday Feb 23, 2022

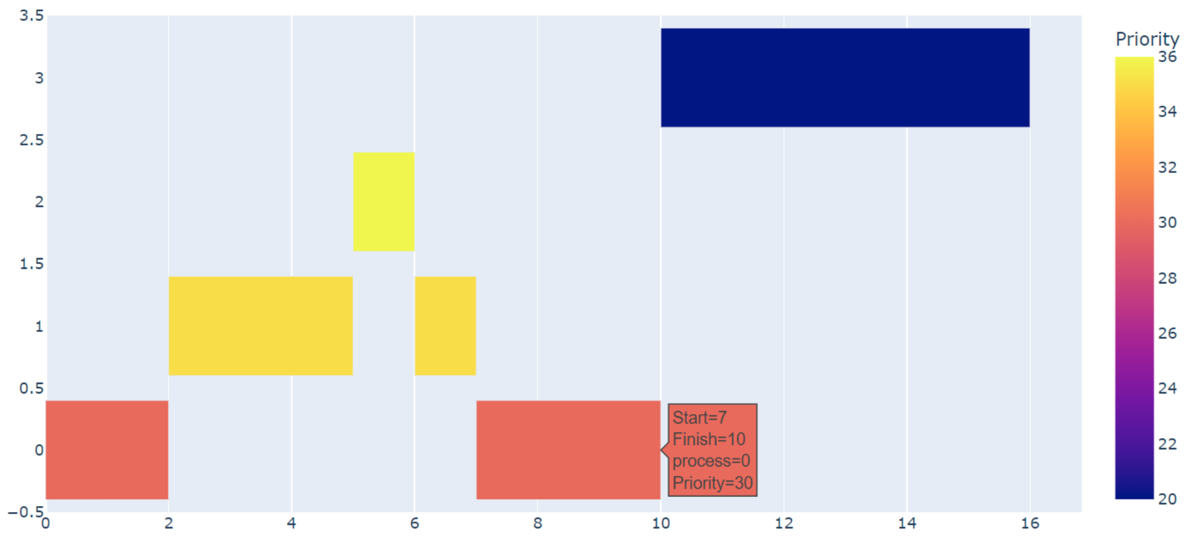


Figure 1: Gantt chart showing PP scheduling simulation of four processes over 16 time slices.

Project objective:

- Understand **preemptive** process scheduling algorithms by implementing them in Python.
- Simulate the CPU scheduling aspects of an Operating System kernel.
- Practice building OS simulations in Python and Jupyter Notebooks.

Project overview:

In this assignment, you will create simulations of the CPU scheduling aspects of an Operating System Kernel and implement the following scheduling algorithms:

1. RR: Round Robin
2. SRT: Shortest Remaining Time
3. PP: Preemptive Priority
4. MFQ: Multilevel Feedback Queue

In addition to the algorithms themselves, you will write code to calculate statistics and simulations of specific scheduling situations that are associated with each algorithm. Finally, you will create a document in Jupyter Notebooks that acts as your **project report** where you run your tests, simulations, and create visualizations of each run. This project is a continuation of the previous project, so we will use the same file structure of *process.py*, *scheduler.py*, *operating_system.py*, and *Scheduling_Analyses2.ipynb*.

The Process Class:

In the process file, you have to make the following modifications to the ***process*** class:

1. Add a new attribute (field) called ***response_time*** and set it to zero in the __*init*__ method. This attribute will be added to the statistics we want to calculate for each process.

2. Replace the process burst time integer with a list called ***duty*** that defines CPU time and I/O time. This list contains integers that alternate between CPU time and I/O time. For example, the following list [5, 2, 7] indicates that this process runs for 5 time slices, then waits for 2 time slices, then runs for 7 time slices. The list always starts with a running time. The full list is only used by Multilevel Feedback Queue, the other three algorithms use only the first number as a burst time. You can modify the first three algorithms to use the full list as an extension.

3. Add an attribute (field) called ***status*** and give it **"running"** as an initial value. This field will alternate between running and waiting according to the process behavior list. When the process is waiting, it should not be in the ready queue.

4. You will need another attribute called ***queue*** to store the current queue number in a Multilevel Feedback Queue.

5. Implement setters and getters for the new attributes.

The scheduler file:

In your ***scheduler.py*** file from the previous project, add the following preemptive algorithms:

1. Round Robin that takes a quantum.
2. Shortest Remaining Time.
3. Preemptive Priority.
4. Multilevel Feedback Queue that consists of three queues:
    a. Round Robin with quantum of 4
    b. Round Robin with quantum of 7
    c. First-Come First Serve (non-preemptive).

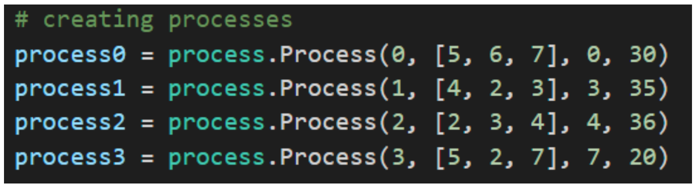
Remember, you can make any modifications to enhance the efficiency, readability, or extendability of the code. General rules of good software design are:

1. Each function should do exactly one thing (single responsibility principle).
2. Splitting a complicated function into multiple simpler functions is desired (functional decomposition).
3. Write good code the first time, never write quick code and plan to come back to it later.

The operating_system file:

The ***operating_system.py*** file remains mostly the same from previous project, in addition to the following:

1. Create a few processes for testing your schedulers, you can use the examples from class to verify your answers. Something like this should be a good starting point:



```
# creating processes
process0 = process.Process(0, [5, 6, 7], 0, 30)
process1 = process.Process(1, [4, 2, 3], 3, 35)
process2 = process.Process(2, [2, 3, 4], 4, 36)
process3 = process.Process(3, [5, 2, 7], 7, 20)
```

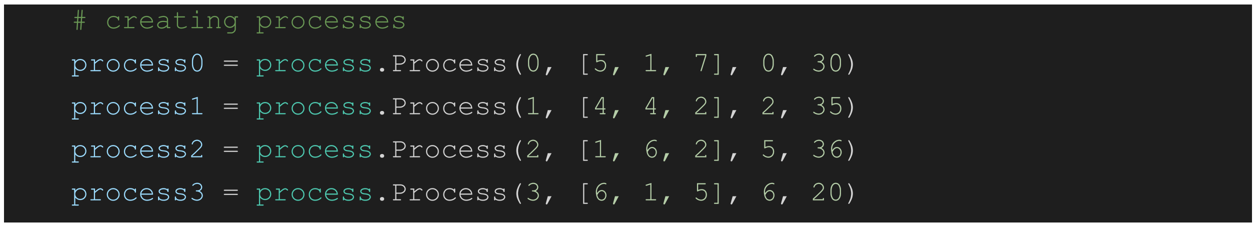
2. Calculate **response-time** in addition to wait-time and turnaround-time for each process, and calculate averages of the three metrics for the entire simulation.

3. Store the results in a pandas dataframe and then store the data frame as a CSV file.

The Scheduling_Analyses2 Jupyter Notebook file:

Create a new notebook and name it ***Scheduling_Analyses2.ipynb***. In this file you will run your tests, visualize the results of each scheduler, run your simulations, and write your report. I will elaborate on each of the three parts separately.

Testing:
Run the **four schedulers** and generate data to verify that the three schedulers work as intended. You can use examples from class to verify your schedulers. For example, the processes below, generate the Gantt chart at Figure 1 with Preemptive Priority scheduling (keeping in mind the base implementation uses only the first CPU-burst in the duty list):



```
    # creating processes
    process0 = process.Process(0, [5, 1, 7], 0, 30)
    process1 = process.Process(1, [4, 4, 2], 2, 35)
    process2 = process.Process(2, [1, 6, 2], 5, 36)
    process3 = process.Process(3, [6, 1, 5], 6, 20)
```

To run a specific scheduler, all you have to do is run the kernel with the name of your scheduler like the example below:

operating_system.kernel(scheduler.PP_scheduler)

Since verbose is kept to the default true value, the line above should generate something like this:

```
process 0        starts: 0 ends: 2
process 1        starts: 2 ends: 5
process 2        starts: 5 ends: 6
process 1        starts: 6 ends: 7
process 0        starts: 7 ends: 10
process 3        starts: 10 ends: 16
```

Visualization: Use the same code from project 1 to generate visualizations for the four algorithms.

Simulations:
Programmatically, generate 1000 processes from which 50/50 split between (use random):

- CPU-bound processes: Processes that spend a lot of time in the CPU, and very little time in I/O. The range of CPU time is between 8 and 12, while I/O is between 1 and 3.
- I/O-bound processes: Processes that spend a lot of time doing I/O, and very little time in the CPU. The range of CPU time is between 1 and 3, while I/O is between 8 and 12.

After you randomize the order of the I/O and CPU bound processes, compare the three metrics (response-time, wait-time, and turnaround-time) when scheduling the generated process list using:

1. Round Robin with quantum of 2
2. Round Robin with quantum of 10
3. Shortest Remaining Time
4. Multilevel Feedback Queue (RR2 => RR 10 => FCFS)

Note: You are allowed to make modifications to your code, including modifying your kernel function to take in a list of processes.

Report:
Your Jupyter Notebook should be a coherent document that includes markdown text and code to communicate the sections above in addition to:

1. Abstract: A brief summary of the project, in your own words. This should be no more than a few sentences. Give the reader context and identify the key purpose of the assignment.
2. Results: A section that goes over schedulers testing, visualization, and simulations.
3. Discussion: A section that interprets and describes the significance of your findings focussing on the simulation results.
4. Extensions: Describe any extensions you undertook, including text output, graphs, tables, or images demonstrating those extensions.
5. References/Acknowledgements.

Extensions:

You can be creative here and come up with your own extensions. I will suggest the following ideas:

- Implement ***aging*** with priority scheduling and show it with an example that would result in starvation without aging (if you have not done so in project 1).
- Come up with your own preemptive scheduling algorithm and explain in which situations it might be useful/better.
- Create elaborate visualizations (if you have not done so in project 1) to show and compare the simulation results (use matplotlib, for example).
- Implement promotion/demotion in Multilevel Feedback Queue.
- Modify the first three schedulers to use the duty list which defines alternating CPU and I/O times.

Project submission:

- Add all your files (except *.ipynb_checkpoints* and *__pycache__*) to your ***project_2*** directory on Google Drive.
- Copy the rubric from Moodle to the project directory after you review it.