BIRZEIT UNIVERSITY

Faculty of Engineering & Technology
Department of Electrical & Computer Engineering
ENCS3390: Operating System Concepts
First Semester, 2024/2025

Programming Task 2

Overview:

In this project, you will simulate a CPU scheduling system combined with deadlock detection and recovery. The system consists of a single CPU core and several resource types, with only one instance of each resource type. The goal is to simulate the execution of processes, apply CPU scheduling, detect deadlocks, and recover from them.

Input format:

Your program will take as input a text file containing a list of processes. Each line in the file represents a process and has the following format:

[PID]   [Arrival Time]   [Priority]   [Sequence of CPU and IO bursts]

check

Each process must start and finish with a CPU burst. Within CPU bursts, processes can request or release resources. For example, the input file might look like this:

0   0   1   **CPU** {R[1], 50, F[1]}

1   5   1   **CPU** {20}   **IO**{30}   **CPU**{20, R[2], 30, F[2], 10}

after, return to ready queue

Explanation:
- **Process 0** (PID = 0) arrives at time 0, has a priority of 1, and consists of one CPU burst. During this burst, it requests resource 1 (R[1]), executes for 50 time units, then releases resource 1 (F[1]) and terminates.
- **Process 1** (PID = 1) arrives at time 5, also with priority 1, and consists of two CPU bursts and one IO burst:
  - The first CPU burst lasts for 20 units, followed by an IO burst lasting 30 units.
  - The second CPU burst executes for 20 units, after which it requests resource 2 (R[2]), executes for 30 units, releases resource 2 (F[2]), and finally finishes with 10 more CPU units.

Your Task:

- **Simulate CPU Scheduling**: Implement priority scheduling algorithm with round robin (see slide 5.27). The simulation should continue until all the processes in the input file terminate.

- **Deadlock Detection and Recovery**: Implement an appropriate deadlock detection algorithm to monitor the system status and identify deadlock situations. If a deadlock is detected, implement a deadlock recovery strategy of your choice (e.g., process termination, resource preemption).

- At the end of simulation, your program must show the Gantt chart representing the timeline of process execution, average waiting time, and average turnaround time for all processes. Also report the detected deadlock states and how recovery was handled.

Remarks:

- **Testing**: Start with simple input files to test your program and ensure its correctness. Gradually test with more complex scenarios.

- **Context Switching**: Assume that the context switch time is negligible and can be ignored in the simulation.

- **I/O Simulation**: When a process finishes its CPU burst, it moves to the I/O queue for the duration of its I/O burst. Processes can perform I/O simultaneously and do not wait for each other. When the I/O burst is finished, the process goes back to the ready queue.'

- **Resource Requests**: If a process requests a resource that is currently held by another process, it should be moved to the **waiting queue** until the resource becomes available. The process will remain in the waiting queue until the requested resource is released by the holding process.

- **Time Representation**: Use a simple time counter for the simulation. The time units do not need to have any real-world meaning, but the simulation should track time correctly.

- **Single-threaded Implementation**: You can implement the simulation in a single-threaded program. If you choose to implement it using **multithreading** or **multiprocessing**, ensure you handle synchronization correctly.

Rules:

- **Groups**: You are allowed to works in groups of two.

- **Output**: You have to submit your code and representative snapshots of outputs. At least one execution scenario with deadlock and recovery, and one snapshot without any deadlock.

- **Deadline**: Sunday 5/1/2025 by midnight.

- **Discussion**: each group will discuss their project with the instructor before a mark is given.