# Process Management Simulation

## Objective:

To simulate process management in an operating system, focusing on process creation, termination, and scheduling. The project includes cycle-based execution, where each burst time represents a single CPU cycle, and the system's state is printed at every cycle.

## Input Format:

The input will be a text file (processes.txt) where each line represents a process with its attributes, including parent-child relationships. (Sample file attached with the project description)

**Example Input File**:

| Process ID | Arrival Time | Burst Time | Priority | Parent ID |
|---|---|---|---|---|
| 1 | 0 | 5 | 3 | 0 |
| 2 | 1 | 3 | 1 | 1 |
| 3 | 2 | 8 | 2 | 0 |
| 4 | 3 | 6 | 4 | 3 |
| 5 | 4 | 2 | 5 | 3 |

In the above file Parent-Child Relationships are:

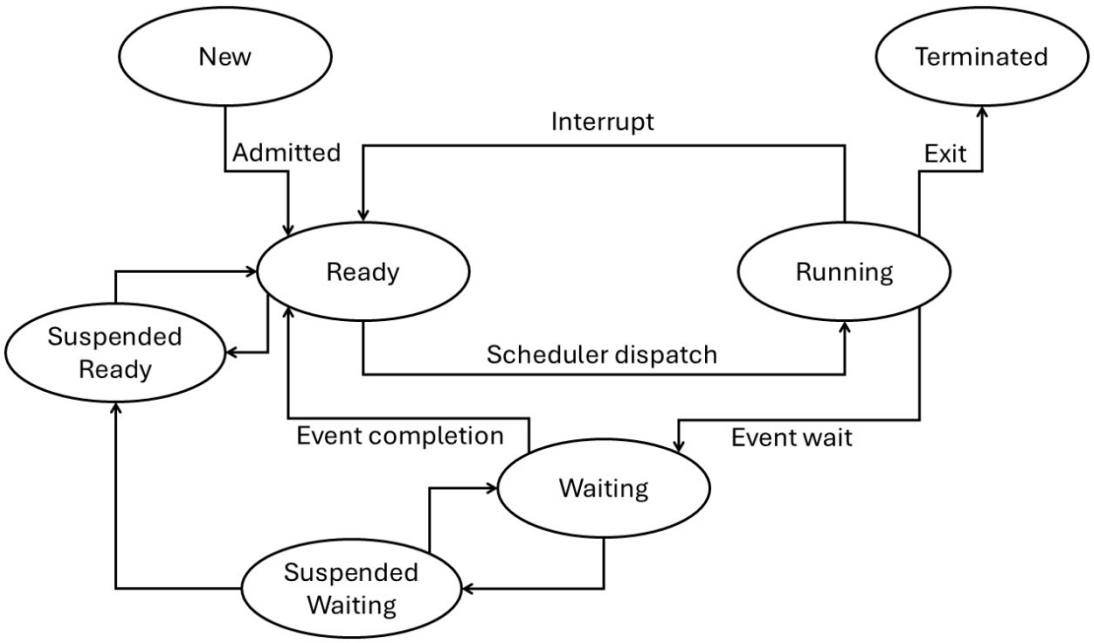
ParentID: P1 -> Children: P2

ParentID: P3 -> Children: P4, P5

ParentID = 0 means its not a Child Process

## Project Requirements:

1. **Process Creation**:
    - Dynamically create parent and child processes from the input file.
    - Allow users to add new or create child processes interactively during runtime (add a prompt after printing out the current cycle).
2. **States:**
    - The processes must follow the transition graph shown below:




    - Note that the ready state can only have 5 processes in it at a time, if you need more than 5, utilize the Suspended Ready state which can handle any number of processes.
    - Same goes for Waiting State, only 5 processes can be in Waiting State at a time, utilize Suspended Waiting State if you need more.
3. **Parent-Child Relationships**:
    - Maintain relationships between parent and child processes.
    - Ensure child processes terminate before the parent terminates.

4. **Cycle-Based Execution**:
    - Treat BurstTime as the number of CPU cycles required for a process.
    - Simulate the execution of processes one cycle at a time.
    - After each cycle, print the **current state** of all processes, including:
        - **Process ID**
        - **Remaining Burst Time**
        - **Status** (Running, Waiting, Terminated)

**Example State (Cycle N)**:

Cycle: N

| Process ID | Remaining Burst Time | Status |
|---|---|---|
| P1 | 4 | Running |
| P2 | 3 | Waiting |
| P3 | 8 | Waiting |
| P4 | 6 | Ready |
| P5 | - | New |

5. **Process Termination**:
    - Simulate termination when a process completes all its cycles.
    - Log the termination of parent and child processes.
6. **Scheduling Algorithms**:
    - Implement the following algorithms when transitioning from Ready state to Running state:
        - First-Come-First-Serve (FCFS)
        - Shortest Job First (SJF)
        - Priority Scheduling
        - Round Robin (RR), Number of cycles per process in Running state is your choice.
    - Once the simulator starts, create a prompt to select between different algorithms.

7. **Performance Metrics**: At the end of simulation calculate and display:
    - Average waiting time.
    - Average turnaround time.
    - Total context switches (Change from one process to another).

**Simulation Outputs:**

1. **Cycle-Wise State**:
    - Print the system's state after each cycle as shown in the example above.
2. **Parent-Child Termination Log**:
    - Log the termination of processes in the correct order, including child processes.
3. **Order of execution**:
    - Visualize the execution order of processes.
    - Example: P1 [5] P2 [8] P3 [16] P4 [22] P5 [24]
    - [] in the above example is denotes the cycle in which the process terminated
4. **Performance Metrics**:
    - Average waiting time.
    - Average turnaround time.
    - Total number of context switches (Change from one process to another).

**Evaluation Criteria:**

1. Correctness of process creation and parent-child management (30%).
2. Accuracy of scheduling algorithm implementations (30%).
3. Correctness of cycle-based execution and state tracking (20%).
4. Code readability, comments, and modularity (10%).
5. Optional features and innovation (10%).