Erich la Torre Boeta Kirsten - Matriculation no.: 94058690

CPU Scheduler Implementation Documentation

1. Core Data Structures

Process Class

```python
@dataclass
class Process:
    pid: int                    # Process ID
    arrival_time: int           # When the process arrives in the system
    burst_time: int             # Total CPU time needed
    priority: int               # Priority level (lower number = higher priority)
    remaining_time: int         # Time left to complete execution
    start_time: Optional[int]   # When process first begins execution
    completion_time: Optional[int] # When process finishes execution
```

The Process class uses Python's @dataclass decorator to automatically generate common methods. It includes two property methods:

- turnaround_time: Calculates total time from arrival to completion
- waiting_time: Calculates total time spent waiting (turnaround time minus burst time)

2. Scheduler Class Components

Initialization and Reset

```python
def __init__(self, processes: List[Process]):
    self.original_processes = processes
    self.reset_processes()
```

- Stores original process list

- Calls reset_processes() to create fresh copies for simulation

```
def reset_processes(self):
    """Reset processes to their initial state for new simulation"""
```

- Creates new Process objects for each simulation
- Ensures each algorithm starts with fresh process states

Metrics Calculation

```
def get_metrics(self) -> Dict:
```

Calculates four key performance metrics:

1. Average Turnaround Time: Mean time from arrival to completion
2. Average Waiting Time: Mean time spent in ready queue
3. CPU Utilization: Percentage of time CPU is busy
4. Throughput: Processes completed per unit time

3. Scheduling Algorithms

First-Come, First-Served (FCFS)

```
def fcfs(self) -> Dict:
```

Implementation steps:

1. Maintains a ready queue of arrived processes
2. Processes are executed in order of arrival
3. No preemption - each process runs to completion
4. Updates start_time when process begins execution
5. Updates completion_time when process finishes

Shortest Job First (SJF)

```
def sjf_nonpreemptive(self) -> Dict:
def sjf_preemptive(self) -> Dict:
```

Non-preemptive version:

1. Similar to FCFS but sorts ready queue by burst_time
2. Once process starts, runs to completion

Preemptive version (Shortest Remaining Time First):

1. Checks for new arrivals after each time unit
2. Can preempt current process if shorter job arrives
3. Sorts ready queue by remaining_time

Round Robin (RR)

```
def round_robin(self, quantum: int = 2) -> Dict:
```

Implementation details:

1. Uses a deque for ready queue (efficient rotation)
2. Each process gets a maximum of quantum time units
3. If process doesn't complete, it goes to back of queue
4. Continues until all processes complete

Priority Scheduling

```
def priority_nonpreemptive(self) -> Dict:
def priority_preemptive(self) -> Dict:
```

Both versions:

1. Sort ready queue by priority value
2. Lower priority number means higher priority

Key differences:

- Non-preemptive: Runs to completion once started

- Preemptive: Can be interrupted by higher priority process

Multilevel Queue

```
def multilevel_queue(self, quantum: int = 2) -> Dict:
```

Implementation features:

1. Uses three priority queues (high, medium, low)
2. High priority queue: First-Come-First-Served
3. Medium/Low priority queues: Round Robin
4. Higher priority queues always execute first
5. Processes are assigned to queues based on priority

4. Process Generation

```
def generate_processes(num_processes: int) -> List[Process]:
```

Creates test processes with:

- Random arrival times (0-50)
- Random burst times (1-20)
- Random priorities (0-10)
- PID assigned sequentially

5. Key Algorithm Features

Time Management

- All algorithms maintain a current_time variable
- Time increases based on:
    - Process execution time
    - Idle time when no processes are ready
    - Time quantum in Round Robin

Queue Management

- Most algorithms use a ready_queue for processes that have arrived
- Some use remaining_processes list for processes yet to arrive
- Multilevel queue uses multiple queues with different policies

Process State Tracking

- start_time: Set when process first begins execution
- completion_time: Set when process finishes
- remaining_time: Updated as process executes
- Various time calculations (turnaround, waiting) derived from these

6. Implementation Notes

Preemption Handling

- Preemptive algorithms check for new arrivals frequently
- Time slicing in Round Robin and Multilevel Queue
- Priority-based interruption in preemptive priority scheduling

Performance Considerations

- Use of deque for efficient queue operations
- Sorting ready queue only when necessary
- Efficient state tracking and updates

Error Handling

- Validates process completion
- Checks for invalid time calculations
- Ensures all processes eventually complete