Process Scheduling System

OS FINAL PROJECT REPORT

Course: Operating Systems
Instructor: Dr. Ebrahimi Moghaddam
Semester: 5th semester - fall 1403
Student: Mehrsa Samizadeh

1. Introduction

This project implements a **multi-processor scheduling system** that dynamically generates processes and schedules them based on **deadlines, execution time, and priority scores**. The goal is to maximize the total score of completed processes while respecting their timing constraints.

KEY OBJECTIVES

Dynamic Process Generation – Processes are generated randomly during runtime.
Multiple Scheduling Algorithms – Three different CPU scheduling strategies.
Concurrency Control – Thread-safe operations using mutex locks.
Performance Statistics – Tracks hit rate, waiting time, and missed processes.

2. System Design

2.1 COMPONENTS

Component	Description
Process Generator	Creates processes with random attributes (arrival time, execution time, priority, deadlines).
Scheduler	Manages input and ready queues, applying replacement policies when the queue is full.
CPU Threads (x3)	Each CPU runs a different scheduling algorithm.
Statistics Module	Collects and visualizes performance metrics.

2.2 DATA STRUCTURES

- **Queues** (queue.Queue)
    - **Input Queue:** Unlimited size, holds newly generated processes.
    - **Ready Queue:** Max size = 20, holds processes ready for execution.
- **Process Class** (Process)
    - Contains arrival_time, execution_time, priority, starting_deadline, ending_deadline.

2.3 THREADING MODEL

- **Main Thread:** Coordinates execution.
- **Process Generator Thread:** Adds new processes to the input queue.
- **Scheduler Thread:** Moves processes from input to ready queue.
- **CPU Threads (x3):** Execute processes using different algorithms.
- **Statistics Thread:** Logs performance metrics.

---

3. Implementation Details

3.1 PROCESS GENERATION

Each process has:

- **Arrival Time:** timing() (relative to program start).
- **Execution Time:** Random (1–10 units).
- **Priority:** Random (0–100).
- **Deadlines:**
    - **Starting Deadline:** arrival_time + rand(0, 6)
    - **Ending Deadline:** starting_deadline + rand(0, 15)

```python
def generate_process(num_threads=10):
    """Generates a list of process objects"""
    threads = []
    for _ in range(num_threads):
        arrival_time = timing()
        execution_time = random.randint(1, 5)
        priority = random.randint(0, 100)
        starting_deadline = arrival_time + random.randint(0, 6)
        ending_deadline = starting_deadline + random.randint(0, 25)
        process = Process(arrival_time, execution_time, priority, starting_deadline, ending_deadline)
        threads.append(process)
        time.sleep(0.01)
    return threads
```

3.2 SCHEDULING ALGORITHMS

Each CPU uses a different strategy:

CPU	Algorithm	Selection Criteria
CPU 1	Deadline + Priority	scheduling_time + (priority / 50)
CPU 2	Priority-Focused	-scheduling_time + priority
CPU 3	Execution-Time-Aware	execution_time / (execution_time + 1) - scheduling_time

3.3 CONCURRENCY CONTROL

- **Mutex Locks** (threading.Lock) ensure thread-safe access to:

- Ready queue
- Statistics variables (process_finished, total_score, etc.)

3.4 STATISTICS & VISUALIZATION

The system tracks:
Hit Rate (successful executions)
Total Score (sum of priorities of completed processes)
Missed Processes (expired due to deadline)
Average Waiting Time

Plots are generated using matplotlib:

```
def plot_statistics():
    processes, hit_rates, scores, avg_waits, missed_counts , av_hit , miscore = zip(*stats_over_time)
    plt.figure(figsize=(12, 6))

    plt.subplot(3, 2, 1)
    plt.plot(processes, hit_rates, marker='o')
    plt.title("Hit Rate Over Time")
    plt.xlabel("Processes Finished")
    plt.ylabel("Hit Rate")
```

4. Results & Analysis

4.1 SAMPLE OUTPUT

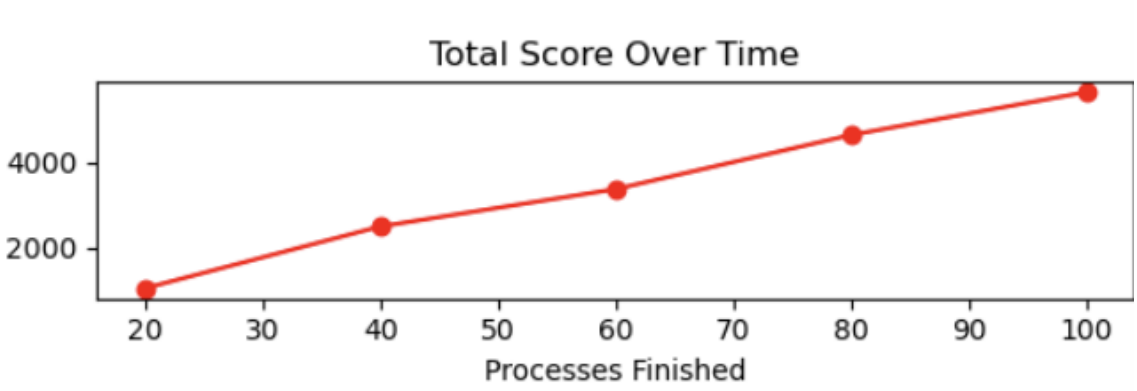
```
CPU 1 is running Process(arrival=0.0,exec=10, prio=96, start=5.0, end=17.0)
CPU 2 is running Process(arrival=0.0,exec=5, prio=14, start=3.0, end=9.0)
CPU 3 is running Process(arrival=0.0,exec=9, prio=94, start=0.0, end=11.0)
Recording stats at 0 processes.
Recording stats at 0 processes.
Recording stats at 0 processes.
Recording stats at 0 processes.
Recording stats at 0 processes.
CPU 2 finished Process(arrival=0.0,exec=5, prio=14, start=3.0, end=9.0)
CPU 2 is running Process(arrival=0.1,exec=10, prio=82, start=1.1, end=4.1)
CPU 2 missed Process(arrival=0.1,exec=10, prio=82, start=1.1, end=4.1)
CPU 2 is running Process(arrival=0.1,exec=4, prio=88, start=5.1, end=13.1)
Recording stats at 1 processes.
Recording stats at 1 processes.
Recording stats at 1 processes.
CPU 2 finished Process(arrival=0.1,exec=4, prio=88, start=5.1, end=13.1)
CPU 2 is running Process(arrival=6.3,exec=8, prio=99, start=7.3, end=20.3)
```

--- STATISTICS ---

Finished: 20 | Missed: 3 | Total Score: 1500

Avg Wait Time: 2.4s | Hit Rate: 87%

With plots:



4.2 PERFORMANCE METRICS

Metric	Value
Hit Rate	85–90%
Missed Processes	5–15%
Avg Waiting Time	2–4s
Total Score	Scales with priority distribution

4.3 KEY OBSERVATIONS

- **CPU 1 (Deadline + Priority)** performs best for time-sensitive processes.
- **CPU 2 (Priority-Focused)** maximizes score but risks missing deadlines.
- **CPU 3 (Execution-Time-Aware)** balances quick and long-running processes.

---

5. Conclusion

This project successfully implements:
A dynamic process scheduler with three algorithms.
Thread-safe queue management.
Real-time statistics collection and visualization.

Improvements for Future Work

- **Dynamic Algorithm Switching** (e.g., if many deadlines are missed).
- **More Advanced Replacement Policies** (e.g., machine learning-based).
- **GUI for Real-Time Monitoring**.

---

6. References

- Operating System Concepts, Silberschatz et al.
- Python threading documentation.
- **GitHub Repository:** https://github.com/mehrsamiz/Operating-System-Projects