CENG 301 Operating Systems
Project Assignment

Process Scheduling Simulator in Python

Assoc. Prof. Ayhan Akbas

Due: 26 December 2025

1. Overview

In this project, you will design and implement a **CPU Process Scheduling Simulator** in **Python 3**. The simulator will support multiple classical CPU scheduling algorithms and will generate detailed logs, statistics, and graphical outputs.

The goal is to deepen your understanding of:

- How an operating system schedules processes on a CPU
- The behaviour and trade-offs of different scheduling algorithms
- Metrics such as waiting time, turnaround time, response time, and context-switch count
- How to design, implement, test, and evaluate a non-trivial system in Python

You must write, run, and debug real code, generate and interpret real outputs, and document your work. You are not allowed to use any AI tool.

2. Learning Outcomes

By the end of this project, you should be able to:

- Implement several CPU scheduling algorithms in Python
- Simulate the execution of processes with arrival times, burst times, and priorities
- Construct Gantt charts and detailed execution logs
- Compute and interpret standard scheduling metrics
- Compare algorithms quantitatively using graphs
- Organise source code in a modular and readable form

3. Programming Language and Libraries

- **Language:** Python 3.
- **Standard library:** You may use any Python standard library modules (e.g. `argparse`, `math`, `itertools`, etc.).
- **Plotting:** You may use **matplotlib** for graphs. Other heavy simulation or plotting frameworks are not allowed.
- **Not allowed:** Any external library or tool that directly implements scheduling simulation for you.

4. Scheduling Algorithms to Implement

You must implement the following CPU scheduling algorithms as separate Python functions/modules:

1. **FCFS** (First-Come First-Served)
2. **SJF (Non-preemptive)** – Shortest Job First
3. **SRTF** (Shortest Remaining Time First) – preemptive version of SJF
4. **Round Robin (RR)** – with configurable time quantum
5. **Priority Scheduling (Non-preemptive)**
6. **Priority Scheduling (Preemptive)**

Each algorithm must:
- Respect process *arrival times*
- Handle ties in a deterministic and documented way (e.g. by PID order)
- Track context switches and execution segments for Gantt chart generation

5. Input Format and Execution

5.1. Input File (`processes.txt`)

Your simulator must accept a text file describing processes. A sample format:

```
# pid  arrival_time  burst_time  priority
P1     0             8           2
P2     1             4           1
P3     2             9           3
P4     3             5           2
```

Rules:
- Lines starting with `#` are comments and must be ignored.
- Fields are separated by whitespace (space or tab).
- `pid` is a string without spaces (e.g. `P1`).
- `arrival_time`, `burst_time`, and `priority` are integers.

5.2. Command-Line Interface

Your main program `scheduler.py` must be runnable from the command line using `argparse`, for example:

```
python scheduler.py --input processes.txt --algo RR --quantum 4
```

Required command-line arguments:
- `--input` *FILENAME*: path to the process description file
- `--algo` *ALGO*: one of `FCFS`, `SJF`, `SRTF`, `RR`, `PRIO_NP`, `PRIO_P`
- `--quantum` Q: time quantum for RR (required if `--algo RR`)

6. Required Outputs

Your simulator must produce several types of output, generated automatically by your Python code.

6.1. ASCII Gantt Chart

Your program must print a readable Gantt chart showing which process runs at which time interval. For example:

```
Time: 0   1   2   3   4   5   6   7   8   9   10
     |---P1---|---P2---|--P1--|--P4--|---P3---|
```

The exact style is flexible, but it must clearly show:

- Time progression
- Which process is running in each interval
- Preemptions and context switches

6.2. Execution Log

Print a detailed timeline log of scheduling events. Example:

```
t=0: P1 arrives, P1 starts running
t=1: P1 running
t=2: P2 arrives
t=4: P2 starts running (preempts P1)
t=6: P3 arrives
...
```

This log should include:

- Process arrivals
- Process start times and preemptions
- Process completions

6.3. Per-Process Statistics

For each process, compute and print the following metrics:

- Completion time
- Turnaround time = Completion time − Arrival time
- Waiting time = Turnaround time − Burst time
- Response time = First start time − Arrival time

Print a formatted table, e.g.:

```
PID  Arr  Burst  Compl  Turnaround  Waiting  Response
P1   0    8      14     14          6        0
P2   1    4      8      7           3        1
P3   2    9      23     21          12       4
P4   3    5      18     15          10       6
```

Also compute and print:

- Average turnaround time
- Average waiting time

- Average response time
- Total number of context switches

6.4. Algorithm Comparison Summary

For a given input workload, run all required algorithms and summarise the results in a comparison table, for example:

Algorithm	Avg. Turnaround	Avg. Waiting	Avg. Response	Context Switches
FCFS	15.5	9.0	2.0	3
SJF	12.0	7.2	1.5	4
SRTF	11.5	6.8	1.0	9
RR(q=4)	13.0	8.3	1.4	11
PRIO_NP	14.2	8.7	1.8	5
PRIO_P	13.7	8.1	1.3	10

The exact numbers will depend on your input, but your program must be able to produce such data.

6.5. Graphs (Using `matplotlib`)

You must generate at least two graphs using Python and save them to files (e.g. in a `graphs/` directory):

a) **Average Waiting Time vs. Algorithm**
b) **Average Turnaround Time vs. Algorithm**

Example usage:

```
import matplotlib.pyplot as plt

# ... after computing metrics ...
plt.bar(algorithms, avg_waiting_times)
plt.xlabel("Algorithm")
plt.ylabel("Average Waiting Time")
plt.title("Average Waiting Time vs Algorithm")
plt.savefig("graphs/waiting.png")
```

7. Project Structure

Your code should be organised in a clean, modular structure. The following is a suggested structure:

```
scheduler/
        scheduler.py
        algorithms/
             fcfs.py
             sjf.py
             srtf.py
             rr.py
             priority_np.py
```

```
        priority_p.py
    utils/
        parser.py
        statistics.py
        gantt.py
    processes.txt
    README.md
```

You are free to adapt this structure, but:

- Each algorithm should be clearly separated.
- Common utilities (parsing, statistics, Gantt chart generation) should not be duplicated.
- Code should be well-commented and readable.

8. Anti-Plagiarism and Authenticity Requirements

Ensure that the project reflects your own work and is not simply generated by an AI tool. You must submit:

1. **Source code** for all Python files, with meaningful comments explaining key parts of the logic.

2. **Execution logs** showing runs of different algorithms on at least one non-trivial input file.

3. **Screenshots**:
   - Terminal output (Gantt charts, tables)
   - The generated graphs (e.g. `waiting.png`, `turnaround.png`)

4. **A short screen-capture video** (approx. 60–90 seconds) demonstrating:
   - How you run the program with different algorithms
   - The output produced
   - A brief verbal explanation of one key observation about the results

Failure to provide these items may result in reduced marks or a suspicion of academic misconduct.

9. Report / README Requirements

You must include at least a **README.md** file that contains:

- Your name, student ID, and section
- Python version and dependencies
- How to run the simulator (sample commands)
- Brief description of each algorithm implementation
- Short discussion of:
  - Which algorithm performed best (and why)
  - Any surprising behaviours you observed

Optionally, you may also prepare a separate PDF report.

10. Grading Rubric

The project will be graded according to the following criteria:

Item	Weight
Correct implementation of required algorithms (FCFS, SJF, SRTF, RR, Priority NP/P)	30%
Correct computation of statistics (waiting, turnaround, response, context switches)	15%
Gantt chart and execution log clarity and correctness	15%
Graph generation (waiting/turnaround vs. algorithm)	15%
Execution logs, screenshots, and demo video authenticity	15%
Code organisation, structure, and comments	5%
README / documentation quality and explanation of results	5%

11. Submission Instructions

Unless otherwise stated, submit:

- A compressed archive (`.zip` or `.tar.gz`) containing:
  - All Python source files
  - `processes.txt` test file(s)
  - Generated graphs (`.png`)
  - Execution logs (as `.txt` or `.md`)
  - Screenshots
  - `README.md`
- Video file

Important: Late submissions are not allowed and will be penalized.