CSC 4320 - Operating Systems
Project 1: Process Scheduling Simulation

Group: Joey Zhang, Phi Ly, Arturo, Sean Pham
Link: GIthub

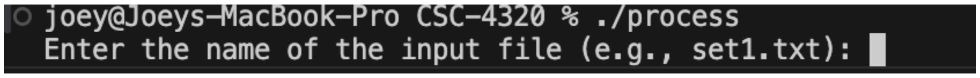
Overview

We decided to implement a simulation for two CPU scheduling algorithms:

1. First-Come, First-Served (FCFS)
2. Shortest Job First (SJF) non-preemptive

The simulation reads process data from a user-specified text file and calculates the **waiting time** (WT) and **turnaround time** (TAT).

Gantt Chart are used to show the execution order for each algorithm. Our program is written in C, and input/test case files are organized into their own folder to support multiple test cases depending on user input upon program prompt.



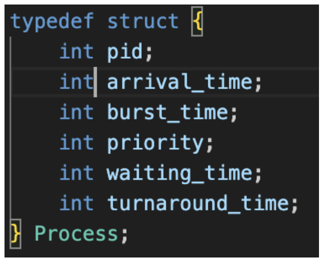
```
joey@Joeys-MacBook-Pro CSC-4320 % ./process
Enter the name of the input file (e.g., set1.txt):
```

Implementation Details

Input Handling

- User is prompted for an input filename
- The file is read from `input/` folder (e.g., `input/set1.txt`)
  - The given text file outlined in the rubric is *processes.txt*
- Each line contains the PID, arrival time, burst time, and priority

Data Structures



```
typedef struct {
    int pid;
    int arrival_time;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
} Process;
```

- A *`Process`* struct stores the process info and performance metrics
- A 2D array `gantt[][]` stores the Gantt chart: `[PID, Start, End]`.

Scheduling Logic

- **FCFS:** Processes are sorted by arrival time and executed in order. If a process arrives after current CPU time, the CPU will idle until the process arrives.
    - Implemented in the function:

      ```
      void fcfs_scheduling(Process processes[], int n, int gantt[][3],
      int* gantt_size)
      ```

    - We use quicksort to sort processes by arrival time:

      ```
      qsort(processes, n, sizeof(Process), compare_arrival);
      ```

    - Then we loop through the sorted list and assign each process a start time `(gantt[*gantt_size][1])` and end time `(gantt[*gantt_size][2])`, accumulating current time with: `time += processes[i].burst_time;`
    - Waiting Time and Turnaround Time are computed using:
        - `processes[i].waiting_time = start_time - arrival_time;`
        - `processes[i].turnaround_time = end_time - arrival_time;`

- **SFJ:** Each scheduling decision, the program will select the shortest job from the list of arrived and unprocessed jobs. If no process has arrived, the CPU idles
    - Implemented in the function:

```
void sjf_scheduling(Process processes[], int n, int gantt[][3], int*
gantt_size)
```

    - We use quicksort again to organize by arrival time
    - Inside the while loop, we search for the shortest burst time linearly using linear scan:

      ```
      for (int i = 0; i < n; i++) {
            if (!done[i] && processes[i].arrival_time <= time) {
            // choose shortest burst time
            }
      }
      ```

    - Completed processes are tracked using a `done[]` array with space allocated set as the defined `MAX_PROCESSES`
    - When no more processes are available (CPU idle), we manually increment time:
    - Once a job is selected, we log it to the Gantt chart and calculate its waiting and turnaround times, just like in FCFS.

      ```
      if (shortest == -1) {
            time++;
            continue;
      }
      ```

Metrics Computed

- **Waiting Time (WT):** `Start Time - Arrival Time`
- **Turnaround Time (TAT):** `Completion Time - Arrival Time`
- **Average** WT and TAT are calculated and printed for comparison

Results

We created multiple input files `set1.txt` to `set5.txt` to test difference scenarios:

- All processes arriving at the same time
- Long process arriving early
- Sparse arrivals with idle CPU time

Summary Table

Test Case	AWT (FCFS)	AWT (SJF)	ATAT (FCFS)	ATAT (SJF)	Winner
set1.txt	10.20	7.40	15.80	13.00	SJF
set2.txt	7.80	4.00	10.80	7.00	SJF
set3.txt	8.20	7.80	12.20	11.80	SJF (slight)
set4.txt	0.20	0.20	4.20	4.20	Tie
set5.txt	4.60	3.60	8.60	7.60	SJF

Observations:

- In set4.txt, all processes had arrival times spaced out such that only one process was ready at a time.
    - In this case, both FCFS and SJF produced the exact same Gantt chart and performance metrics, because no scheduling decision had to be made — each process was run as it arrived.
- In set1.txt, set2.txt, set3.txt, and set5.txt, SJF significantly reduced average waiting time (AWT) and average turnaround time (ATAT).
    - The improvements were most noticeable when shorter jobs arrived later in the queue
- When a longer process arrives first, as in set3.txt, FCFS makes all shorter jobs wait — causing high average waiting time

set1.txt

FCFS Gantt Chart:
| P1 | P2 | P3 | P4 | P5 |
0 8 12 21 26 28

SJF Gantt Chart:
| P1 | P2 | P5 | P4 | P3 |
0 8 12 14 19 28**Avg WT:** FCFS: 10.20 | SJF: 7.40
Avg TAT: FCFS: 15.80 | SJF: 13.00

PID	FCFS WT	FCFS TAT	SJF WT	SJF TAT
1	0	8	0	8
2	7	11	7	11
3	10	19	17	26
4	18	23	11	16
5	16	18	2	4

set2.txt

FCFS Gantt Chart:
| P1 | P2 | P3 | P4 | P5 |
0 5 8 12 14 15

SJF Gantt Chart:
| P5 | P4 | P2 | P3 | P1 |
0 1 3 6 10 15

Avg WT: FCFS: 7.80 | SJF: 4.00
Avg TAT: FCFS: 10.80 | SJF: 7.00

PID	FCFS WT	FCFS TAT	SJF WT	SJF TAT
1	0	5	10	15
2	5	8	3	6
3	8	12	6	10
4	12	14	1	3
5	14	15	0	1

set3.txt

FCFS Gantt Chart:
| P1 | P2 | P3 | P4 | P5 |
0 12 14 15 18 20

SJF Gantt Chart:
| P1 | P3 | P2 | P5 | P4 |
0 12 13 15 17 20

Avg WT: FCFS: 8.20 | SJF: 7.80

Avg TAT: FCFS: 12.20 | SJF: 11.80

PID	FCFS WT	FCFS TAT	SJF WT	SJF TAT
1	0	12	0	12
2	9	11	10	12
3	10	11	8	9
4	10	13	12	15
5	12	14	9	11

set4.txt

Gantt Chart (Both FCFS & SJF):
| P1 | P2 | P3 | P4 | P5 |
5 9 13 17 26 31**Avg WT:** 0.20
Avg TAT: 4.20

Note: FCFS and SJF gave identical results due to non-overlapping arrival times.

PID	WT	TAT
1	0	4
2	0	3
3	0	2
4	0	6
5	1	6

set5.txt

FCFS Gantt Chart:
| P1 | P2 | P3 | P4 | P5 |
0 3 9 13 18 20

SJF Gantt Chart:
| P1 | P2 | P5 | P3 | P4 |
0 3 9 11 15 20

Avg WT: FCFS: 4.60 | SJF: 3.60
Avg TAT: FCFS: 8.60 | SJF: 7.60

PID	FCFS WT	FCFS TAT	SJF WT	SJF TAT
1	0	3	0	3
2	1	7	1	7
3	5	9	7	11
4	7	12	9	14
5	10	12	1	3

Challenges & Solutions

One of the first challenges is struggling with the basics of C, still continues to be a challenge as there are still things that still could be probably be tweaked:

Handling Large Process Files

- **Challenge:** Max processes is fixed at 100, if the input is at 100 the other processes running is ignored

Error Checking

- **Challenge:** If user prints wrong filename, it prints an error and doesn't allow retrying
- Solution could be just putting a loop on the code so you can redo it again

SJF Idle Time

- **Challenge:** If all the processes haven't arrived yet, it increments the time
- **Solution:** Have it jump to the next earliest process

Bugs for sorting in SJF

- **Challenge:** It should pick the next shortest available job at each step.
- **Solution:** Have it find the next shortest burst time, should be in a loop.

Handling Idle CPU Time

- **Challenge:** In the early versions of our SJF implementation, the CPU kept getting stuck when no processes had arrived by current time.
- **Solution:** This was fixed by checking whether a valid process was found. If not, we incremented current_time++ and continued scanning, which simulates the actual idle behavior of the CPU

Gantt Chart

- **Challenge:** Occasionally misaligned or showed wrong time intervals from incorrect start/end time calculations
- **Solution:** We had to make sure that each scheduled process, the start time = max(current_time, arrival_time), and then increment by burst time. We also had to double check that the output formatting was correct for clarity but that was easy after referencing the official documentation for print formatting.

FIle Input Bugs

- **Challenge:** We had improper formatting of the .txt input files which kept causing failed reads or incorrect data
- **Solution:** In the code we made sure to always skip the header using get() and just made sure that all input lines were formatted consistently with 4 integer values: PID Arrival_Time Burst_Time Priority. Finally we tested each file one by one to debug.

Our Full Output Results:

```
joey@Joeys-MacBook-Pro Project1_Scheduling % ./process
Enter the name of the input file (e.g., set1.txt): set1.txt

=== FCFS Scheduling ===
Gantt Chart:
| P1 | P2 | P3 | P4 | P5 |
0   8   12   21   26   28

PID      WT       TAT
1        0        8
2        7        11
3        10       19
4        18       23
5        16       18

Avg WT: 10.20
Avg TAT: 15.80

=== SJF Scheduling ===
Gantt Chart:
| P1 | P2 | P5 | P4 | P3 |
0   8   12   14   19   28

PID      WT       TAT
1        0        8
2        7        11
3        17       26
4        11       16
5        2        4

Avg WT: 7.40
Avg TAT: 13.00
joey@Joeys-MacBook-Pro Project1_Scheduling % ./process
Enter the name of the input file (e.g., set1.txt): set2.txt

=== FCFS Scheduling ===
Gantt Chart:
| P1 | P2 | P3 | P4 | P5 |
0   5   8   12   14   15

PID      WT       TAT
1        0        5
2        5        8
3        8        12
4        12       14
5        14       15

Avg WT: 7.80
Avg TAT: 10.80

=== SJF Scheduling ===
Gantt Chart:
| P5 | P4 | P2 | P3 | P1 |
0   1   3   6   10   15

PID      WT       TAT
1        10       15
2        3        6
3        6        10
4        1        3
5        0        1

Avg WT: 4.00
Avg TAT: 7.00
joey@Joeys-MacBook-Pro Project1_Scheduling % ./process
```

```
Enter the name of the input file (e.g., set1.txt): set3.txt

=== FCFS Scheduling ===
Gantt Chart:
| P1 | P2 | P3 | P4 | P5 |
0   12   14   15   18   20

PID      WT       TAT
1        0        12
2        9        11
3        10       11
4        10       13
5        12       14

Avg WT: 8.20
Avg TAT: 12.20

=== SJF Scheduling ===
Gantt Chart:
| P1 | P3 | P2 | P5 | P4 |
0   12   13   15   17   20

PID      WT       TAT
1        0        12
2        10       12
3        8        9
4        12       15
5        9        11

Avg WT: 7.80
Avg TAT: 11.80
joey@Joeys-MacBook-Pro Project1_Scheduling % ./process
Enter the name of the input file (e.g., set1.txt): set4.txt

=== FCFS Scheduling ===
Gantt Chart:
| P1 | P2 | P3 | P4 | P5 |
5   9   13   17   26   31

PID      WT       TAT
1        0        4
2        0        3
3        0        2
4        0        6
5        1        6

Avg WT: 0.20
Avg TAT: 4.20

=== SJF Scheduling ===
Gantt Chart:
| P1 | P2 | P3 | P4 | P5 |
5   9   13   17   26   31

PID      WT       TAT
1        0        4
2        0        3
3        0        2
4        0        6
5        1        6

Avg WT: 0.20
Avg TAT: 4.20
joey@Joeys-MacBook-Pro Project1_Scheduling % ./process
Enter the name of the input file (e.g., set1.txt): set5.txt

=== FCFS Scheduling ===
Gantt Chart:
| P1 | P2 | P3 | P4 | P5 |
0   3   9   13   18   20

PID      WT       TAT
1        0        3
2        1        7
3        5        9
4        7        12
5        10       12

Avg WT: 4.60
Avg TAT: 8.60

=== SJF Scheduling ===
Gantt Chart:
| P1 | P2 | P5 | P3 | P4 |
0   3   9   11   15   20

PID      WT       TAT
1        0        3
2        1        7
3        7        11
4        9        14
5        1        3

Avg WT: 3.60
Avg TAT: 7.60
```