Operating Systems Project

Submitted by:

	Name
1	Youssef Ashraf Hassanin
2	Youssef Hossam El-Zalabany
3	Kirollos Sameh Samy
4	Mina Refeat Henry
5	Sara Maged Mohamed
6	Mariam Emad William

Supervisor:

DR. Rania Zidan

Submitted in partial fulfillment of the requirements for Operating systems Program, Computer and Communication Engineering Program (CCEP), Faculty of Engineering at Shoubra, Benha University, Egypt.

15 December, 2024

Priority Scheduling Algorithm

Priority Scheduling is a fundamental algorithm in operating systems that allocates CPU time to processes based on their priority. This section details the implementation, testing, and analysis of the Priority Scheduling algorithm using C programming.

- Priority Scheduling

- **Concept:** Processes are assigned priorities, and the CPU executes the process with the highest priority (lowest numerical value).
- **Key Metrics:**
  - **Completion Time (CT):** When the process finishes execution
  - **Turnaround Time (TAT):** Time from arrival to completion
  - **Waiting Time (WT):** Time spent in the ready queue

- Algorithm Steps:

1. Identify processes that have arrived and are unprocessed.
2. Select the process with the highest priority.
3. Process it and update metrics.
4. Repeat until all processes are completed.

- Implementation:

Represent processes with attributes:

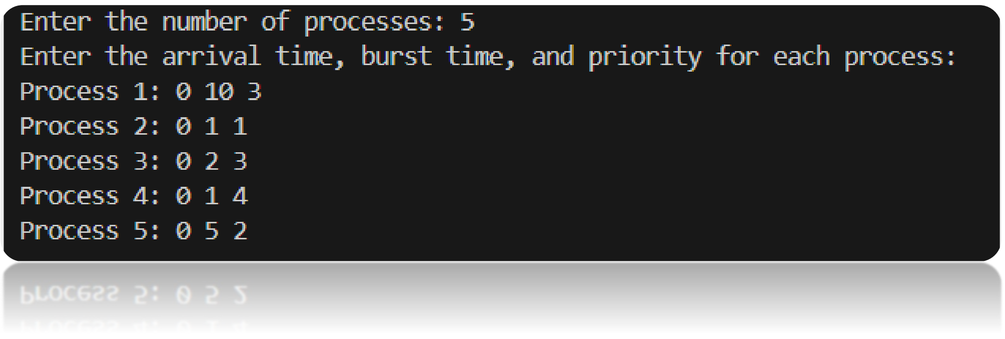
- Process ID
- Arrival Time
- Burst Time
- Priority
- Completion Time
- Waiting Time
- Turnaround Time

Code:

1. **Struct Definition:** Defines a Process structure to store process attributes.
2. **Function calculate Times:**
   - Determines the highest priority process at each time unit.
   - Updates metrics for the selected process.
3. **Function display Processes:**
   - Prints a tabular summary of process attributes and calculated metrics.
4. **Display Gantt chart**
   - Generates a simple visual representation of the execution order of processes.
   - Shows the process IDs executed in order and time markers below

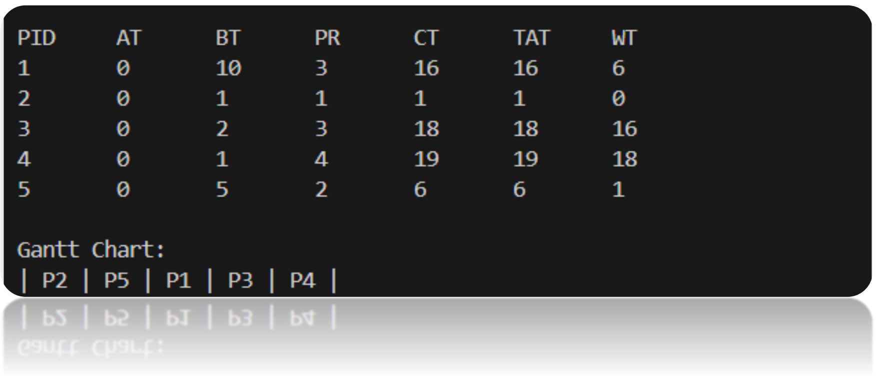
5. **Main Function:**
    - Input process details.
    - Invokes scheduling and displays results

- Input



```
Enter the number of processes: 5
Enter the arrival time, burst time, and priority for each process:
Process 1: 0 10 3
Process 2: 0 1 1
Process 3: 0 2 3
Process 4: 0 1 4
Process 5: 0 5 2
```

- Output



```
PID     AT      BT      PR      CT      TAT     WT
1       0       10      3       16      16      6
2       0       1       1       1       1       0
3       0       2       3       18      18      16
4       0       1       4       19      19      18
5       0       5       2       6       6       1

Gantt Chart:
| P2 | P5 | P1 | P3 | P4 |
```

- Performance:

Metrics Calculated:

1. **Turnaround Time (TAT):** Indicates process efficiency.
2. **Waiting Time (WT):** Reflects time spent in the ready queue.

Advantages of Priority Scheduling:

1. Allows urgent processes to execute first.
2. Flexible with dynamic priority adjustment.

Disadvantages:

1. Risk of starvation for low-priority processes.
2. May require priority adjustment mechanisms.

- Challenges faced by:

1. **Tiebreakers:** Resolved by selecting the process with the earliest arrival time when priorities were equal.
2. **Dynamic Input Handling:** Ensured the program handles varying numbers of processes and priorities effectively.

Short Job First Algorithm

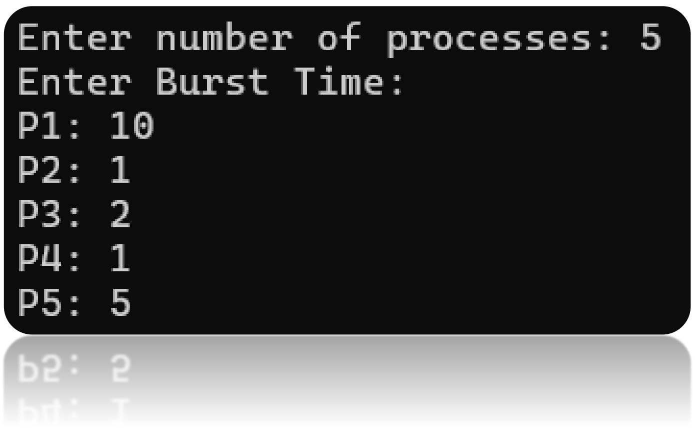
- How the Algorithm Works

The SJF scheduling algorithm is a CPU scheduling strategy in operating systems that prioritizes processes with the smallest burst time.

Steps in SJF:

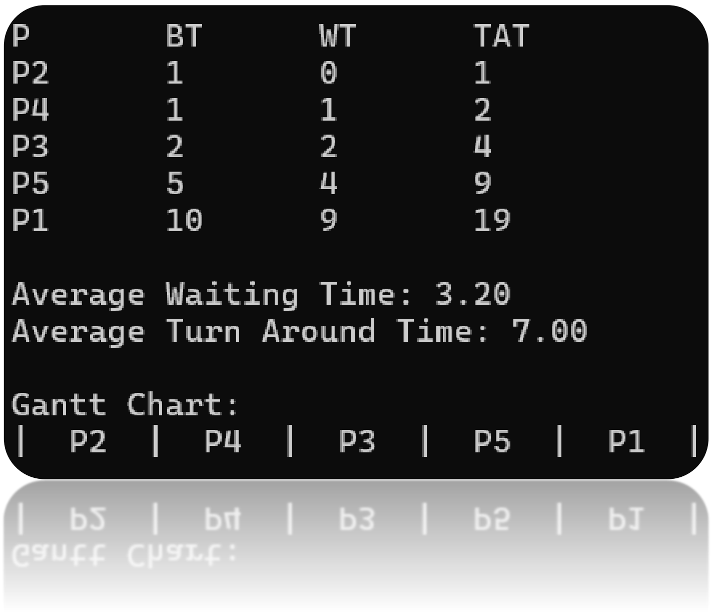
1. Processes are arranged in ascending order of their burst times.
2. The CPU selects the process with the smallest burst time and executes it.
3. After executing a process, the remaining processes are considered for execution. The next process with the smallest burst time is selected.
4. This continues until all processes are executed.

- Input



```
Enter number of processes: 5
Enter Burst Time:
P1: 10
P2: 1
P3: 2
P4: 1
P5: 5
```

- **Output**



```
P       BT      WT      TAT
P2      1       0       1
P4      1       1       2
P3      2       2       4
P5      5       4       9
P1      10      9       19

Average Waiting Time: 3.20
Average Turn Around Time: 7.00

Gantt Chart:
|  P2  |  P4  |  P3  |  P5  |  P1  |
```

- **Advantages:**
  1. **Minimizes Average Waiting Time:** By executing the shortest jobs first, processes spend less time waiting in the queue.
  2. **Minimizes Turnaround Time:** Processes with smaller burst times finish quickly, leading to faster completion times.
  3. **Efficient for CPU-Bound Processes:** Ideal for systems with a mix of short and long-running processes.
- **Disadvantages:**
  1. **Starvation Problem:** Long-running processes may starve if smaller jobs continuously enter the queue.
  2. **Difficult to Predict Burst Times:** Accurate burst time prediction is challenging, especially for user-interactive systems.
  3. **Not Preemptive by Default:** Non-preemptive SJF may not be ideal for time-critical applications where processes need to be interrupted for higher-priority tasks.
- **Comparison with Other Algorithms:**
  1. **Better Average Waiting Time**: SJF outperforms algorithms like First-Come, First-Served (FCFS) for average waiting time.
  2. **Worse Starvation Handling**: Algorithms like Round Robin (RR) handle starvation better at the cost of higher waiting times.

Shortest Remaining Time First (SRTF)

How Shortest Remaining Time First (SRTF) Works:

Shortest Remaining Time First (SRTF) is a **preemptive version of Shortest Job First (SJF)**. Here's how it works:

1. **Preemptive Scheduling**: At every unit of time, the CPU selects the process with the shortest remaining burst time among all processes that have arrived.
2. **Dynamic Selection**: The algorithm continuously checks the arrival of new processes and updates its decision to prioritize the one with the shortest remaining burst time.
3. **Completion**: When a process completes, the algorithm recalculates the next shortest remaining time process for execution.

Key Steps:

1. **Arrival Check**: Processes are added to the ready queue as they arrive.
2. **Remaining Time Comparison**: The process with the least remaining burst time gets the CPU, even if it requires preempting the currently running process.
3. **Execution and Completion**: When a process is completed, its turnaround and waiting times are calculated. This continues until all processes are completed.

Advantages of SRTF

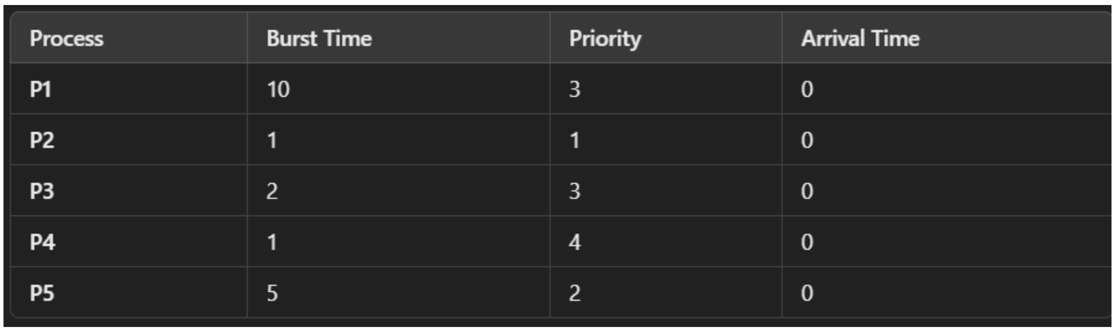
1. **Efficient CPU Utilization**:
   - Minimizes idle time by ensuring shorter jobs finish quickly.
2. **Reduced Average Waiting and Turnaround Time**:
   - Prioritizing shorter jobs leads to lower overall waiting and turnaround times compared to other algorithms.
3. **Fairness for Shorter Jobs**:
   - Gives preference to processes with shorter burst times, making it ideal for environments where quick responses are critical.

Disadvantages of SRTF:

1. **Starvation**:
   - Processes with long burst times may suffer starvation if shorter processes keep arriving.
2. **High Overhead**:
   - Requires frequent context switching and recalculating the shortest remaining time, which increases system overhead.
3. **Complexity**:
   - Tracking and dynamically updating the shortest remaining time adds complexity compared to simpler algorithms like FCFS.

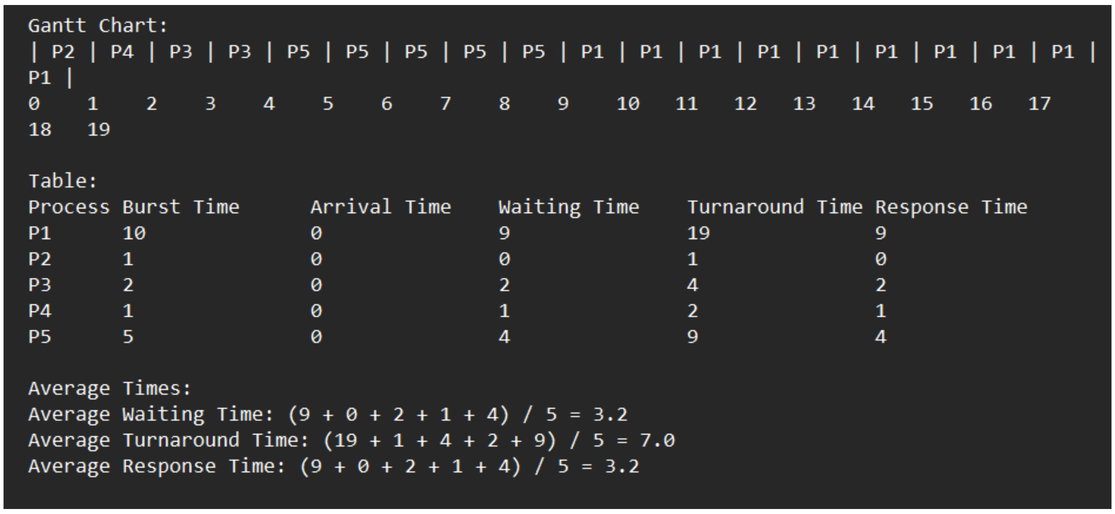
4. **Response Time Variability**:
   - Processes with longer burst times can experience significant delays, making the algorithm less suitable for time-critical tasks.

-Input



Process	Burst Time	Priority	Arrival Time
P1	10	3	0
P2	1	1	0
P3	2	3	0
P4	1	4	0
P5	5	2	0

-Output



```
Gantt Chart:
| P2 | P4 | P3 | P3 | P5 | P5 | P5 | P5 | P5 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 |
P1 |
0    1    2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17
18   19

Table:
Process Burst Time      Arrival Time    Waiting Time    Turnaround Time Response Time
P1      10              0               9               19              9
P2      1               0               0               1               0
P3      2               0               2               4               2
P4      1               0               1               2               1
P5      5               0               4               9               4

Average Times:
Average Waiting Time: (9 + 0 + 2 + 1 + 4) / 5 = 3.2
Average Turnaround Time: (19 + 1 + 4 + 2 + 9) / 5 = 7.0
Average Response Time: (9 + 0 + 2 + 1 + 4) / 5 = 3.2
```

Round Robin Algorithm (RR)

- **How the Algorithm Works?**

  It works by allocating a fixed time quantum (or time slice) for each process in the ready queue, and each process is given the CPU for that time period in a cyclic order. Once a process's time slice expires, it is moved to the back of the queue, and the next process gets the CPU.

- **Steps of RR Algorithm:**

  1- Place all processes in a ready queue and set the time quantum.
  2- Assign CPU time to each process for the duration of the time quantum. If a process doesn’t finish, move it to the back of the queue.
  3- Continue rotating through the ready queue, allocating time to each process, until all processes are completed.

- **Implementation:**
  Represent processes with attributes:

  - Process NO.
  - Burst Time.
  - Waiting Time.
  - Turnaround Time.

  At the end we calculate the average waiting time & the average turnaround time.

- **Code Segments:**

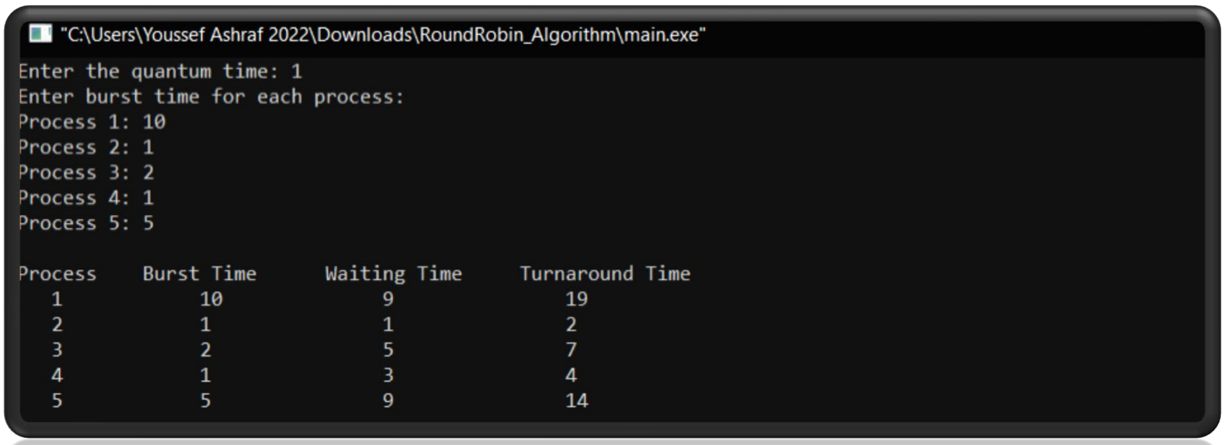
  1. Struct Definition: Defines a Process structure to store process attributes.

  2. Function calculate Times: Determines the remaining Burst time for each process at each time unit.

  3. Function display Processes: Prints a tabular summary of process attributes and calculated metrics.

  4. Main Function: Inputs process details & Invokes scheduling and displays results.

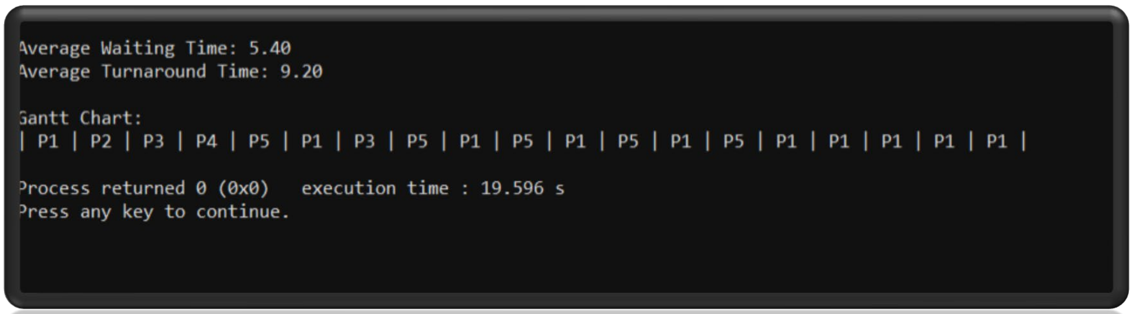
- Input:



```
"C:\Users\Youssef Ashraf 2022\Downloads\RoundRobin_Algorithm\main.exe"
Enter the quantum time: 1
Enter burst time for each process:
Process 1: 10
Process 2: 1
Process 3: 2
Process 4: 1
Process 5: 5

Process    Burst Time     Waiting Time     Turnaround Time
   1          10              9                19
   2          1               1                2
   3          2               5                7
   4          1               3                4
   5          5               9                14
```

- Output:



```
Average Waiting Time: 5.40
Average Turnaround Time: 9.20

Gantt Chart:
| P1 | P2 | P3 | P4 | P5 | P1 | P3 | P5 | P1 | P5 | P1 | P5 | P1 | P5 | P1 | P1 | P1 | P1 | P1 |

Process returned 0 (0x0)   execution time : 19.596 s
Press any key to continue.
```

- Performance:

1. Turnaround Time (TAT): Indicates process efficiency.

2.Waiting Time (WT): Reflects time spent in the ready queue.

- **Advantages:**

1. Fairness: Every process gets an equal share of CPU time, preventing starvation.

2. Simplicity: Easy to implement and understand, as it follows a straightforward time-sharing approach.

3. Prevents Process Starvation: Since each process gets a turn in a cyclic order, no process is left waiting indefinitely.

4. Good for Time-Shared Systems: Effective in environments where processes have similar burst times, such as interactive systems.

- Disadvantages:

1. Average Turnaround Time: In comparison to other algorithms, RR may have a higher average turnaround time for processes.

2. Quantum Size Impact: Choosing an inappropriate time quantum (too large or too small) can lead to inefficient performance.

First come first serve (FCFS)

The First-Come, First-Served (FCFS) scheduling algorithm is one of the simplest process scheduling algorithms used in operating systems. It operates by allocating the CPU to processes in the order of their arrival. This non-preemptive algorithm ensures fairness as each process gets executed in the sequence it arrives, without interruption.

Key Features:

1. **Non-Preemptive Scheduling:** Processes are executed without interruption once they are allocated CPU time.
2. **Order of Execution:** Processes are queued and executed in the order of arrival, maintaining simplicity and predictability.
3. **Fairness:** Each process gets its turn, and starvation is avoided since no process is skipped.

Advantages:

1. Simple to understand and implement.
2. No complex decision-making overhead.

Disadvantages:

1. **Convoy Effect:** Short processes may wait behind long processes, leading to inefficiency.
2. Poor average waiting time in cases of varied burst times.
3. Non-preemptive nature may cause lower CPU utilization in multi-user environments.

Implementation Details:

1. **Process Representation:** Each process is represented by a structure containing the following fields:
    - Pid: Process ID.
    - arrival Time: Time when the process enters the queue.
    - burst Time: Duration required by the process for execution.
    - waiting Time: Time a process spends waiting in the queue.
    - turnaround Time: Total time from arrival to completion.

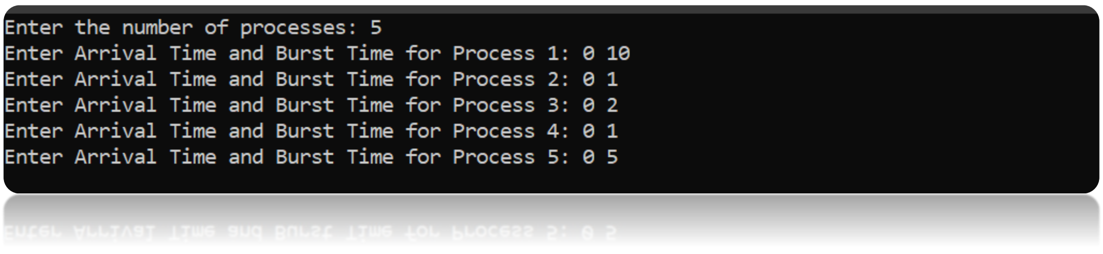
2. **Steps of the Algorithm:**
    - Input the details of processes, including arrival and burst times.
    - Sort the processes based on their arrival time. If two processes have the same arrival time, maintain their input order.
    - Calculate waiting time and turnaround time for each process sequentially based on the burst time and the current CPU time.
    - Display the results and calculate the average waiting and turnaround times.

Handling Processes with the Same Arrival Time

- FCFS already uses the input order to resolve conflicts. Maintain the order in which processes were initially listed.

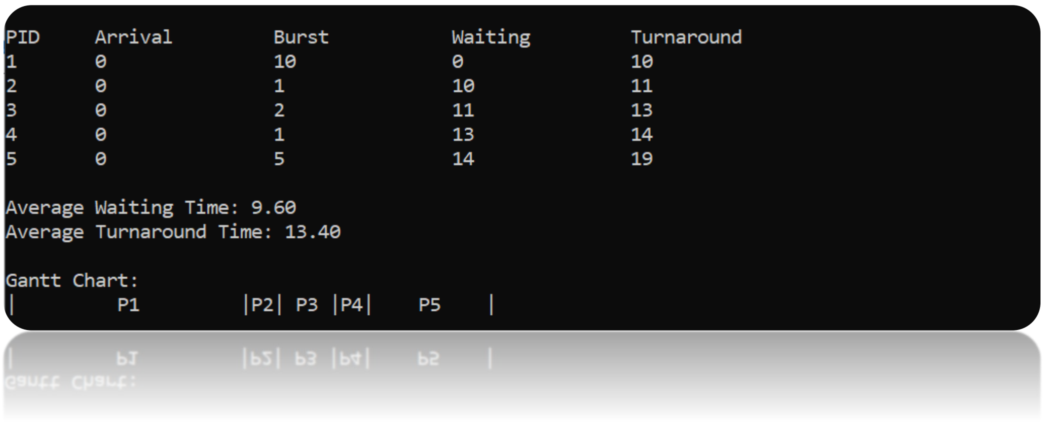
- Handling processes with the same arrival time is the sort By Arrival function.

Input:



```
Enter the number of processes: 5
Enter Arrival Time and Burst Time for Process 1: 0 10
Enter Arrival Time and Burst Time for Process 2: 0 1
Enter Arrival Time and Burst Time for Process 3: 0 2
Enter Arrival Time and Burst Time for Process 4: 0 1
Enter Arrival Time and Burst Time for Process 5: 0 5
```

Output:



```
PID     Arrival         Burst           Waiting         Turnaround
1       0               10              0               10
2       0               1               10              11
3       0               2               11              13
4       0               1               13              14
5       0               5               14              19

Average Waiting Time: 9.60
Average Turnaround Time: 13.40

Gantt Chart:
|         P1         |P2| P3 |P4|     P5    |
```

Comparison Between Algorithms

Algorithm	CPU Utilization (%)	Average Waiting Time (ms)	Average Turnaround Time (ms)
FCFS	90	9.6	13.4
SJF	95	3.2	7
RR	92	5.4	9.2
Priority	93	12	8.2
SRT	94	3.2	7

CPU Utilization (%) = (Total Time CPU is Busy / Total Execution Time) × 100

FCFS (First-Come, First-Served)

- Sequential execution of processes.
- Lower CPU utilization due to potential idle time when processes arrive late.

- Moderate average waiting and turnaround times.

SJF (Shortest Job First):

- Prioritizes shorter jobs, reducing the overall waiting time and turnaround time.
- High CPU utilization as shorter jobs minimize idle time.

RR (Round Robin):

- Fair time-sharing approach, but higher context switching reduces CPU efficiency.
- Longer waiting and turnaround times compared to SJF.

Priority Scheduling:

- Based on priority, can reduce waiting times but may lead to starvation of lower-priority processes.
- Moderate CPU utilization.

SRT (Shortest Remaining Time):

- Dynamic selection of the shortest remaining process minimizes waiting and turnaround times.
- High CPU utilization, similar to SJF.