DAYANANDA SAGAR ACADEMY OF TECHNOLOGY & MANAGEMENT
Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore- 560082
(Affiliated to Visvesvaraya Technological University, Belagavi and Approved by AICTE, New Delhi)

NAAC Accredited with Grade A+
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
Accredited by NBA, New Delhi
(Autonomous Batch under VTU)

Continuous and Comprehensive Activity On

Operating Systems (BCS304)

Academic Year: **2025-26**

Execution of Scheduling Algorithms

Name : Shridhar Vinodsa Solanki
USN : 1DT25CS459
Semester: 3 Section: F

Handling Faculty: Ms Manusha A

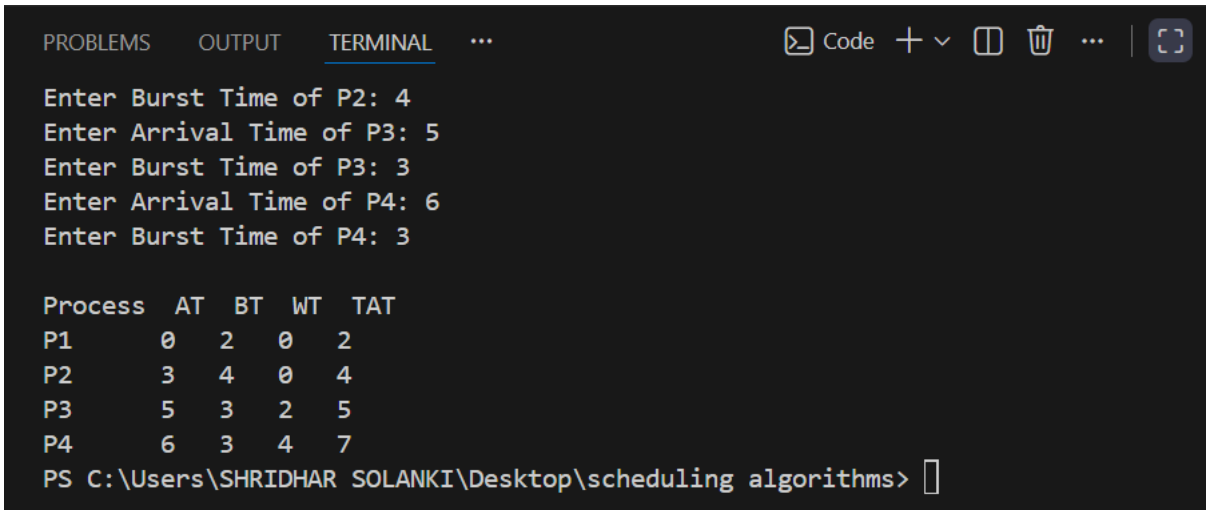
Index

First Come First Serve (FCFS) :-

First Come First Serve (FCFS) is one of the simplest CPU scheduling algorithms used in operating systems. In this method, processes are executed strictly in the order in which they arrive in the ready queue. The scheduling follows the First In First Out (FIFO) principle, meaning the process that enters the system first gets access to the CPU first. Once a process is allocated the CPU, it continues execution until its burst time is completed. FCFS is a non-preemptive scheduling algorithm and is easy to understand and implement, but it may result in higher waiting time when long processes are executed before shorter ones.

fcfs.py > ...

```python
n = int(input("Enter number of processes: "))
processes = []

for i in range(n):
    at = int(input(f"Enter Arrival Time of P{i+1}: "))
    bt = int(input(f"Enter Burst Time of P{i+1}: "))
    processes.append((i+1, at, bt))

processes.sort(key=lambda x: x[1])

time = 0
wt = {}
tat = {}

for p, at, bt in processes:
    if time < at:
        time = at
    wt[p] = time - at
    time += bt
    tat[p] = wt[p] + bt

print("\nProcess  AT  BT  WT  TAT")
for p, at, bt in processes:
    print(f"P{p}      {at}   {bt}   {wt[p]}   {tat[p]}")

```



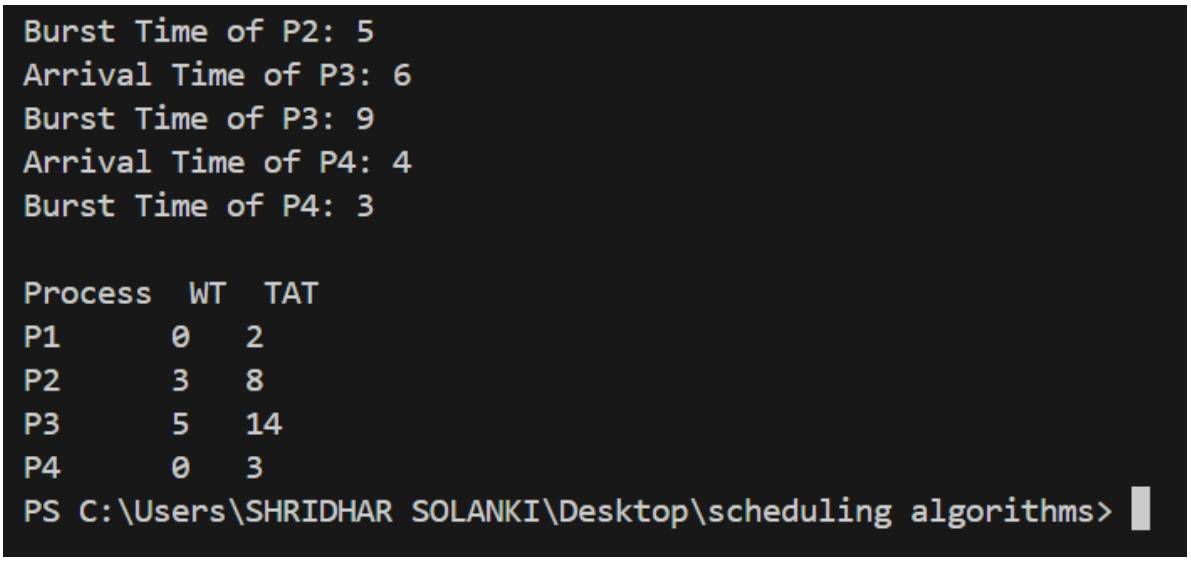
Applications: FCFS scheduling is commonly used in batch operating systems where jobs are processed sequentially without user interaction. It is also applied in printer spooling systems, where print jobs are handled in the order they are received. FCFS can be seen in customer service systems such as banks and ticket counters, where people are served on a first-come basis. It is suitable for simple scheduling environments where fairness is more important than performance.

2. Shortest Remaining Time First (SRTF)

Shortest Remaining Time First (SRTF) is a preemptive CPU scheduling algorithm and is considered an extension of the Shortest Job First scheduling technique. In this algorithm, the CPU is always assigned to the process that has the smallest remaining execution time. If a new process arrives with a shorter remaining burst time than the currently running process, the CPU preempts the current process and executes the new one. This approach helps in reducing the average waiting time and improves system responsiveness. However, frequent preemption can increase overhead and may cause starvation of longer processes.

srtf.py > ...

```python
n = int(input("Enter number of processes: "))
at = []
bt = []
rt = []

for i in range(n):
    at.append(int(input(f"Arrival Time of P{i+1}: ")))
    b = int(input(f"Burst Time of P{i+1}: "))
    bt.append(b)
    rt.append(b)

time = 0
completed = 0
wt = [0]*n

while completed < n:
    idx = -1
    min_rt = 10**9
    for i in range(n):
        if at[i] <= time and rt[i] > 0 and rt[i] < min_rt:
            min_rt = rt[i]
            idx = i

    if idx == -1:
        time += 1
        continue

    rt[idx] -= 1
    time += 1

    if rt[idx] == 0:
        completed += 1
        wt[idx] = time - at[idx] - bt[idx]

tat = [wt[i] + bt[i] for i in range(n)]

print("\nProcess  WT  TAT")
for i in range(n):
    print(f"P{i+1}      {wt[i]}   {tat[i]}")
```



Applications:

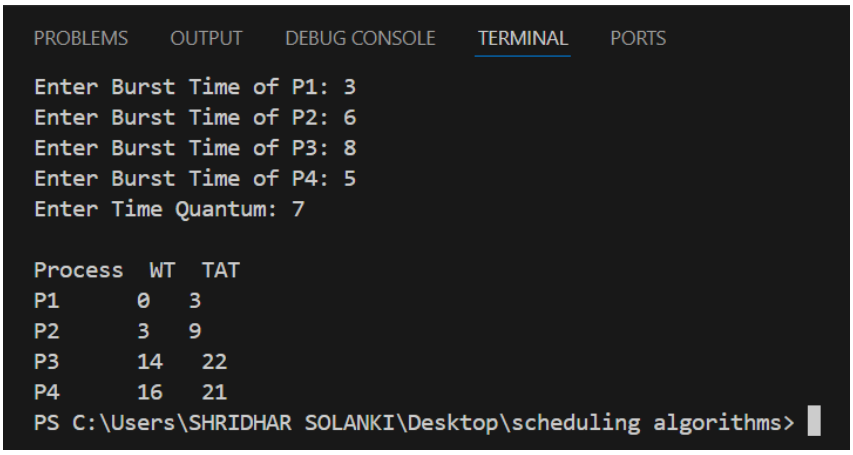
SRTF scheduling is used in interactive operating systems where quick response time is important. It is suitable for time-sharing systems that handle multiple user requests simultaneously. This algorithm is also useful in real-time systems where short and critical tasks need immediate attention. SRTF is preferred in environments where minimizing average waiting time is a key requirement.

3. Round Robin (RR)

Round Robin is a CPU scheduling algorithm mainly designed for time-sharing operating systems. In this algorithm, each process is assigned a fixed time slice known as the time quantum. Processes are executed in a use the CPU. If a process does not complete its execution within the allotted time quantum, it is preempted and moved to the end of the ready queue. This process continues until all processes are completed. Round Robin improves fairness and prevents starvation, but the performance depends heavily on the selection of an appropriate time quantum.

round robin.py > ...

```python
n = int(input("Enter number of processes: "))
bt = []
rt = []

for i in range(n):
    burst = int(input(f"Enter Burst Time of P{i+1}: "))
    bt.append(burst)
    rt.append(burst)

tq = int(input("Enter Time Quantum: "))
time = 0
wt = [0]*n

while True:
    done = True
    for i in range(n):
        if rt[i] > 0:
            done = False
            if rt[i] > tq:
                time += tq
                rt[i] -= tq
            else:
                time += rt[i]
                wt[i] = time - bt[i]
                rt[i] = 0
    if done:
        break

tat = [wt[i] + bt[i] for i in range(n)]

print("\nProcess  WT  TAT")
for i in range(n):
    print(f"P{i+1}      {wt[i]}   {tat[i]}")
```



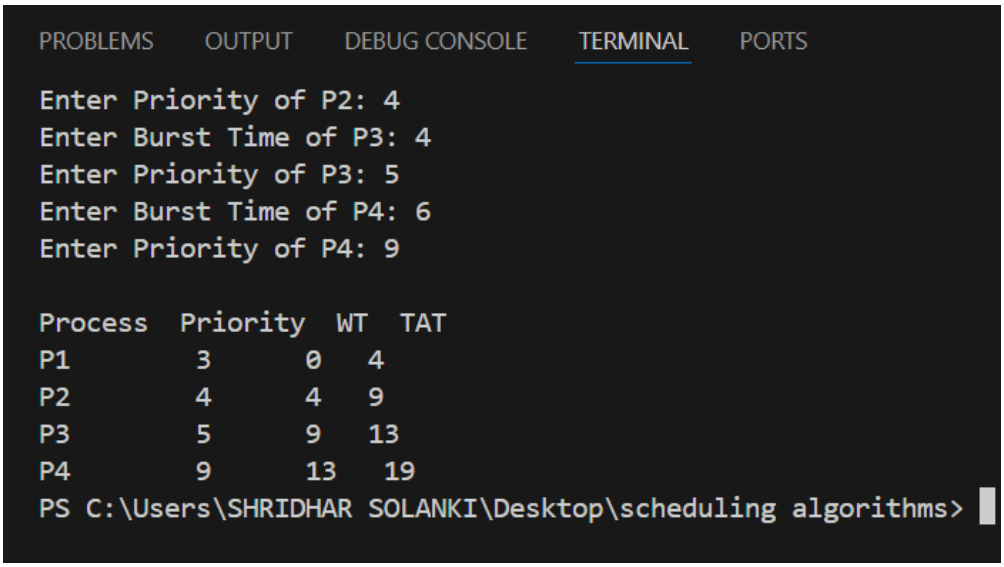
Applications: Round Robin scheduling is widely used in time-sharing operating systems where multiple users interact with the system simultaneously. It is applied in multitasking environments to ensure fair CPU distribution among processes. Round Robin is commonly used in web servers that handle many client requests at the same time. It is also suitable for interactive applications such as text editors and command-line interfaces where responsiveness is important.

4. Priority Scheduling (Non-Preemptive)

Priority Scheduling is a CPU scheduling algorithm in which each process is assigned a priority value based on its importance. The CPU always selects the process with the highest priority for execution. In most systems, a lower priority number indicates a higher priority. In non-preemptive priority scheduling, once a process starts execution, it continues until completion, even if a higher-priority process arrives later. This algorithm ensures that critical processes are executed first, but it may lead to starvation of low-priority processes if high-priority tasks continue to arrive.

priority.py > ...

```python
n = int(input("Enter number of processes: "))
processes = []

for i in range(n):
    bt = int(input(f"Enter Burst Time of P{i+1}: "))
    pr = int(input(f"Enter Priority of P{i+1}: "))
    processes.append((i+1, bt, pr))

processes.sort(key=lambda x: x[2])

wt = [0]*n
tat = [0]*n

for i in range(1, n):
    wt[i] = wt[i-1] + processes[i-1][1]

for i in range(n):
    tat[i] = wt[i] + processes[i][1]

print("\nProcess  Priority  WT  TAT")
for i in range(n):
    p, bt, pr = processes[i]
    print(f"P{p}        {pr}      {wt[i]}   {tat[i]}")

```



Applications: Priority scheduling is commonly used in real-time operating systems where certain tasks must be executed immediately. It is applied in medical systems and emergency services where critical operations cannot be delayed. Industrial automation systems use priority scheduling to handle important control tasks efficiently. This algorithm is also suitable for embedded systems where task importance varies based on system requirements.