VIDHI ROHIRA
SY B.TECH
COMPUTER ENGINEERING

231071052

OS LAB 3

LAB 3

AIM:- To implement CPU Scheduling Algorithms.

THEORY:-

1] <u>First-Come-First-Serve</u>:-

What is FCFS?

First Come First Serve (FCFS) is a scheduling algorithm that executes processes in the order they arrive in the ready queue, without preemption.

Advantages and Disadvantages:

- **Advantage**: Simple and easy to implement; fair in terms of arrival order.
- **Disadvantage**: Can lead to high average waiting time and poor response time, especially with long processes (convoy effect).

Use:

Commonly used in batch processing systems and simple scheduling environments where timing is less critical.

2] <u>Shortest-Job-First</u>:-

1. **What is SJF?**
   Shortest Job First (SJF) is a scheduling algorithm that selects the process with the smallest execution time to run next, minimizing overall waiting time.
2. **Advantages and Disadvantages**:
   - **Advantage**: Reduces average waiting time and turnaround time, making it efficient for short processes.
   - **Disadvantage**: Can lead to starvation for longer processes and requires knowledge of process durations in advance.
3. **Use**:
   Often used in environments where process durations are predictable, such as in batch systems and real-time systems.

3] Round-Robin-Shceduling:-

What is RR?
Round Robin is a preemptive scheduling algorithm that assigns a fixed time quantum to each process in the ready queue, allowing them to execute in a cyclic order.

Advantages and Disadvantages:

- **Advantage**: Fair allocation of CPU time; responsive for time-sharing systems, preventing starvation.
- **Disadvantage**: Performance can degrade with very short time quanta due to context switching overhead.

Use:
Commonly used in time-sharing systems and multi-user environments, where responsiveness is important.

4] <u>Priority-Scheduling</u>:-

What is Priority Scheduling?
Priority Scheduling assigns CPU time to processes based on their priority level, with higher priority processes being executed before lower priority ones.

Advantages and Disadvantages:

- **Advantage**: Efficient for time-sensitive tasks; important processes can be prioritized, reducing response time for critical applications.
- **Disadvantage**: Can lead to starvation of lower-priority processes and may require complex management of priorities.

Use:
Commonly used in real-time systems and environments where process importance varies, such as operating systems and embedded systems.

<u>SOME IMPORTANT TERMS</u>:-

Arrival Time: The time at which a process arrives in the ready queue for execution.

Completion Time: The time at which a process finishes execution and leaves the system.

Burst Time: The total time required by a process for execution on the CPU.

Turnaround Time: The total time taken from arrival to completion, including waiting and execution time.

Waiting Time: The total time a process spends in the ready queue waiting for CPU time, excluding its burst time.

ALGORITHMS:-

1] FCFS:-

Input the Number of Processes (`n`) Along with Their:

- Process IDs.
- Arrival times.
- Burst times.

Sort the Processes:

- Sort based on arrival time in ascending order.

Initialize Variables:

- `currentTime = 0`
- `totalWaitingTime = 0`
- `totalTurnaroundTime = 0`

For Each Process:

- If `currentTime < arrivalTime`, set `currentTime = arrivalTime`.
- Calculate waiting time:
  - `waitingTime = currentTime - arrivalTime`
- Calculate turnaround time:
  - `turnaroundTime = waitingTime + burstTime`
- Calculate completion time:
  - `completionTime = currentTime + burstTime`
- Update `currentTime`:
  - `currentTime += burstTime`
- Accumulate the total waiting and turnaround times.

Calculate Averages:

- Average waiting time:
  - `avgWaitingTime = totalWaitingTime / n`
- Average turnaround time:
  - `avgTurnaroundTime = totalTurnaroundTime / n`

Display:

- Output the process table with arrival time, burst time, completion time, waiting time, and turnaround time.
- Output the average waiting and turnaround times.

2] SJF

Input Process Details:

- Read the number of processes (`n`).
- For each process, input:
  - Process ID.
  - Arrival time: The time at which the process arrives in the ready queue.
  - Burst time: The amount of CPU time required to execute the process.

Initialize Variables:

- `currentTime = 0`: Keeps track of the current system time.
- `completed = 0`: A counter to track how many processes have completed execution.
- `totalWaitingTime = 0`: Accumulates the total waiting time of all processes.
- `totalTurnaroundTime = 0`: Accumulates the total turnaround time of all processes.
- `isCompleted[]`: A boolean array initialized to `false` for all processes to indicate whether a process has finished execution.

While All Processes Are Not Completed (`completed != n`):

- Find the process with the shortest burst time:
  - Loop through all processes and select the process that:
    - Has arrived (`arrivalTime <= currentTime`).
    - Has not yet been completed (`isCompleted[i] == false`).
    - Has the smallest burst time.

    - If multiple processes have the same burst time, the first one found is selected.
- If no process meets the criteria (i.e., no process has arrived or all arrived processes are completed):
    - Increment `currentTime` to simulate system idle time until a new process arrives.

For the Selected Process (with the Shortest Burst Time):

- Calculate waiting time:
    - Waiting time = `currentTime - arrivalTime`.
- Calculate turnaround time:
    - Turnaround time = Waiting time + Burst time.
- Calculate completion time:
    - Completion time = `currentTime + burstTime`.
- Update the `currentTime`:
    - `currentTime += burstTime` (move the clock forward by the burst time of the selected process).
- Mark the process as completed:
    - `isCompleted[idx] = true`.
- Increment the `completed` counter:
    - `completed++`.
- Add the waiting and turnaround times to the totals:
    - `totalWaitingTime += waitingTime`.
    - `totalTurnaroundTime += turnaroundTime`.

Repeat Steps 3 and 4 Until All Processes Have Been Completed (`completed == n`).

Calculate Averages:

- Average waiting time:
    - `avgWaitingTime = totalWaitingTime / n`.
- Average turnaround time:
    - `avgTurnaroundTime = totalTurnaroundTime / n`.

Display the Results:

- For each process, display:

    - Process ID, arrival time, burst time, completion time, turnaround time, and waiting time.
- Display the average waiting time and average turnaround time.

3] RR

1. Input Process Details:

- Read the number of processes `n`.
- For each process `P[i]`, input:
    - `pid[i]`: Process ID.
    - `arrivalTime[i]`: Arrival time.
    - `burstTime[i]`: Burst time.
- Input the time quantum `TQ`.

2. Initialize Variables:

- Set `currentTime = 0`, `completed = 0`.
- Initialize arrays for `waitingTime[]`, `turnaroundTime[]`, `completionTime[]`, and `remainingTime[]` (initially equals `burstTime[]`).
- Create a `readyQueue` for processes ready for execution.
- Sort processes by arrival time.

3. Process Execution:

- While not all processes are completed (`completed != n`):
    1. Select Process: Pop from the front of `readyQueue`.
    2. Execute Process:
        - If remaining time > `TQ`, execute for `TQ`, decrement remaining time, and increment `currentTime`.
        - Else, execute to completion, set remaining time to 0, and update `completionTime[i]`.

        - Increment `completed` and calculate `turnaroundTime[i]` and `waitingTime[i]`.
    3. **Check for New Arrivals**: Add processes with `arrivalTime <= currentTime` and remaining time > 0 to `readyQueue`.
    4. **Re-add Current Process**: If not completed, re-add it to `readyQueue`.
    5. **Handle Idle Time**: If `readyQueue` is empty, set `currentTime` to the next process's arrival time.

4. Calculate Average Times:

- Average Waiting Time: `avgWaitingTime = sum(waitingTime[]) / n`.
- Average Turnaround Time: `avgTurnaroundTime = sum(turnaroundTime[]) / n`.

5. Output Results:

- Display for each process: Process ID, arrival time, burst time, completion time, turnaround time, and waiting time.
- Display average waiting and turnaround times.

4] PRIORITY SCHEDULING:-

1. Input Process Details:

- Read the number of processes `n`.
- For each process `P[i]`, input:
    - `pid[i]`: Process ID.
    - `arrivalTime[i]`: Arrival time.
    - `burstTime[i]`: Burst time.
    - `priority[i]`: Priority (lower number = higher priority).

2. Sort Processes:

- Sort by priority (lower number = higher priority).
- If priorities are the same, use arrival time to break ties.

3. Initialize Variables:

- Set `currentTime = 0`.
- Initialize `totalWaitingTime = 0` and `totalTurnaroundTime = 0`.

4. Calculate Waiting and Turnaround Times:

- For each sorted process `P[i]`:
    1. Update Current Time: If `currentTime < arrivalTime[i]`, set `currentTime = arrivalTime[i]`.
    2. Calculate Waiting Time: `waitingTime[i] = currentTime - arrivalTime[i]`.
    3. Calculate Turnaround Time: `turnaroundTime[i] = waitingTime[i] + burstTime[i]`.
    4. Update Current Time: `currentTime += burstTime[i]`.
    5. Update Totals: Add `waitingTime[i]` to `totalWaitingTime` and `turnaroundTime[i]` to `totalTurnaroundTime`.

5. Calculate Average Times:

- Average Waiting Time: `avgWaitingTime = totalWaitingTime / n`.
- Average Turnaround Time: `avgTurnaroundTime = totalTurnaroundTime / n`.

6. Output Results:

- Display each process's details: Process ID, arrival time, burst time, waiting time, turnaround time, and priority.
- Display average waiting time and average turnaround time.

CODE's AND OUTPUT's:-

FCFS

```
class FCFS:

    def __init__(self, name, arrival_time, burst_time):

        self.name = name

        self.arrival_time = arrival_time

        self.burst_time = burst_time

        self.waiting_time = 0

        self.turnaround_time = 0


def find_waiting_time(processes):

    n = len(processes)

    waiting_time = [0] * n

    waiting_time[0] = 0  # The first process has no waiting time


    for i in range(1, n):

        waiting_time[i] = (processes[i-1].burst_time + waiting_time[i-1])


    return waiting_time


def find_turnaround_time(processes, waiting_time):

    n = len(processes)

    turnaround_time = [0] * n


    for i in range(n):

        turnaround_time[i] = processes[i].burst_time + waiting_time[i]
```

```python
    return turnaround_time


def find_average_time(processes):
    waiting_time = find_waiting_time(processes)
    turnaround_time = find_turnaround_time(processes, waiting_time)


    total_waiting_time = sum(waiting_time)
    total_turnaround_time = sum(turnaround_time)


    n = len(processes)
    print("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time")
    for i in range(n):

print(f"{processes[i].name}\t{processes[i].arrival_time}\t\t{processes[i].burst_time}\t\t{waiting_time[i]}\t\t{turnaround_time[i]}")


    print(f"\nAverage Waiting Time: {total_waiting_time / n:.2f}")
    print(f"Average Turnaround Time: {total_turnaround_time / n:.2f}")
# Example usage
if __name__ == "__main__":
    processes = [
        FCFS("P1", 0, 4),
        FCFS("P2", 1, 3),
        FCFS("P3", 2, 1),
        FCFS("P4", 3, 2)
    ]
    find_average_time(processes)
```

OUTPUT:-

```
/bin/python3.10 /home/vjti/Desktop/CE_52/FCFS.py
vjti@vjti-OptiPlex-3050-AIO:~/Desktop/CE_52$ /bin/python3.10 /home/vjti/Desktop/CE_52/FCFS.py
Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
P1      0               4               0               4
P2      1               3               4               7
P3      2               1               7               8
P4      3               2               8               10

Average Waiting Time: 4.75
Average Turnaround Time: 7.25
vjti@vjti-OptiPlex-3050-AIO:~/Desktop/CE_52$
```

bash
Python

SJF

```python
class SJF:
    def __init__(self, name, arrival_time, burst_time):
        self.name = name
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.waiting_time = 0
        self.turnaround_time = 0

def find_sjf(processes):
    # Sort processes based on arrival time and then by burst time
    processes.sort(key=lambda x: (x.arrival_time, x.burst_time))

    n = len(processes)
    completed = [False] * n
    current_time = 0
    total_waiting_time = 0
```

```python
    total_turnaround_time = 0

    for _ in range(n):
        # Find the process with the smallest burst time that has arrived
        idx = -1
        min_burst = float('inf')

        for i in range(n):
            if not completed[i] and processes[i].arrival_time <= current_time:
                if processes[i].burst_time < min_burst:
                    min_burst = processes[i].burst_time
                    idx = i

        if idx != -1:
            # Process the selected job
            completed[idx] = True
            current_time += processes[idx].burst_time
            processes[idx].waiting_time = current_time - processes[idx].arrival_time - processes[idx].burst_time
            processes[idx].turnaround_time = current_time - processes[idx].arrival_time

            total_waiting_time += processes[idx].waiting_time
            total_turnaround_time += processes[idx].turnaround_time
        else:
            # If no process is available, increment time
            current_time += 1
```

```
    return total_waiting_time, total_turnaround_time


def display_results(processes, total_waiting_time, total_turnaround_time):
    print("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time")
    for p in processes:
        print(f"{p.name}\t{p.arrival_time}\t\t{p.burst_time}\t\t{p.waiting_time}\t\t{p.turnaround_time}")

    n = len(processes)
    print(f"\nAverage Waiting Time: {total_waiting_time / n:.2f}")
    print(f"Average Turnaround Time: {total_turnaround_time / n:.2f}")


# Example usage
if __name__ == "__main__":
    processes = [
        SJF("P1", 0, 8),
        SJF("P2", 1, 4),
        SJF("P3", 2, 9),
        SJF("P4", 3, 5)
    ]

    total_waiting_time, total_turnaround_time = find_sjf(processes)
    display_results(processes, total_waiting_time, total_turnaround_time)
```

OUTPUT:-

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER
vjti@vjti-OptiPlex-3050-AIO:~/Desktop/CE_52$ /bin/python3.10 /home/vjti/Desktop/CE_52/SJF.py
Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
P1      0               8               0               8
P2      1               4               7               11
P3      2               9               15              24
P4      3               5               9               14

Average Waiting Time: 7.75
Average Turnaround Time: 14.25
vjti@vjti-OptiPlex-3050-AIO:~/Desktop/CE_52$
```

RR

```python
class RR:
    def __init__(self, name, arrival_time, burst_time):
        self.name = name
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.remaining_time = burst_time
        self.waiting_time = 0
        self.turnaround_time = 0

def round_robin(processes, quantum):
    time = 0
    queue = []
    n = len(processes)

    while True:
        # Add all arrived processes to the queue
        for process in processes:
```

```python
            if process.arrival_time <= time and process.remaining_time > 0
and process not in queue:
                queue.append(process)

        if not queue:  # If no process is ready
            time += 1
            continue

        # Get the first process in the queue
        current_process = queue.pop(0)

        # Process for the quantum time or remaining time
        if current_process.remaining_time > quantum:
            time += quantum
            current_process.remaining_time -= quantum
            queue.append(current_process)  # Re-queue the process
        else:
            time += current_process.remaining_time
            current_process.waiting_time = time -
current_process.arrival_time - current_process.burst_time
            current_process.turnaround_time = time -
current_process.arrival_time
            current_process.remaining_time = 0  # Process is finished

        # Check if all processes are completed
        if all(process.remaining_time == 0 for process in processes):
            break

    return processes
```

```python
def display_results(processes):
   print("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time")
   total_waiting_time = 0
   total_turnaround_time = 0

   for p in processes:
print(f"{p.name}\t{p.arrival_time}\t\t{p.burst_time}\t\t{p.waiting_time}\t\t{p.turnaround_time}")
       total_waiting_time += p.waiting_time
       total_turnaround_time += p.turnaround_time

   n = len(processes)
   print(f"\nAverage Waiting Time: {total_waiting_time / n:.2f}")
   print(f"Average Turnaround Time: {total_turnaround_time / n:.2f}")

# Example usage
if __name__ == "__main__":
   processes = [
       RR("P1", 0, 8),
       RR("P2", 1, 4),
       RR("P3", 2, 9),
       RR("P4", 3, 5)
   ]
     quantum_time = 3  # Define the quantum time
   completed_processes = round_robin(processes, quantum_time)
   display_results(completed_processes)
```

OUTPUT:-

```
vjti@vjti-OptiPlex-3050-AIO:~/Desktop/CE_52$ /bin/python3.10 /home/vjti/Desktop/CE_52/RR.py
Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
P1      0               8               9               17
P2      1               4               13              17
P3      2               9               15              24
P4      3               5               15              20

Average Waiting Time: 13.00
Average Turnaround Time: 19.50
vjti@vjti-OptiPlex-3050-AIO:~/Desktop/CE_52$
```

PRIORITY SCHEDULING:-

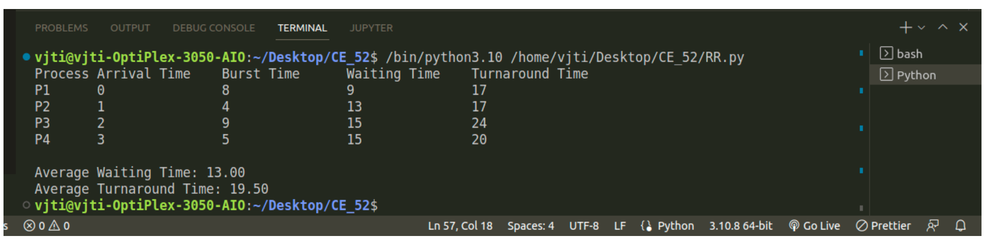
```python
class PS:
    def __init__(self, name, arrival_time, burst_time, priority):
        self.name = name
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.priority = priority
        self.waiting_time = 0
        self.turnaround_time = 0

def priority_scheduling(processes):
    # Sort processes by arrival time and then by priority
    processes.sort(key=lambda x: (x.arrival_time, x.priority))

    n = len(processes)
    completed = 0
```

```
    current_time = 0

    while completed < n:
        # Get the list of processes that have arrived and are not completed
        available_processes = [p for p in processes if p.arrival_time <=
current_time]

        # Filter out completed processes
        available_processes = [p for p in available_processes if
p.burst_time > 0]

        if available_processes:
            # Sort available processes by priority (lower number = higher
priority)
            available_processes.sort(key=lambda x: x.priority)
            current_process = available_processes[0]

            # Process the current process
            current_time += current_process.burst_time
            current_process.waiting_time = current_time -
current_process.arrival_time - current_process.burst_time
            current_process.turnaround_time = current_time -
current_process.arrival_time
            completed += 1
        else:
```

```
            # If no process is available, increment time
            current_time += 1

    return processes


def display_results(processes):
    print("Process\tArrival Time\tBurst Time\tPriority\tWaiting
Time\tTurnaround Time")
    total_waiting_time = 0
    total_turnaround_time = 0

    for p in processes:

print(f"{p.name}\t{p.arrival_time}\t\t{p.burst_time}\t\t{p.priority}\t\t{p
.waiting_time}\t\t{p.turnaround_time}")
        total_waiting_time += p.waiting_time
        total_turnaround_time += p.turnaround_time

    n = len(processes)
    print(f"\nAverage Waiting Time: {total_waiting_time / n:.2f}")
    print(f"Average Turnaround Time: {total_turnaround_time / n:.2f}")


# Example usage
if __name__ == "__main__":
```

```python
processes = [
    PS("P1", 0, 8, 2),
    PS("P2", 1, 4, 1),
    PS("P3", 2, 9, 3),
    PS("P4", 3, 5, 4)
]
completed_processes = priority_scheduling(processes)
display_results(completed_processes)
```

OUTPUT:-



```
vjti@vjti-OptiPlex-3050-AIO:~/Desktop/CE_52$ /bin/python3.10 /home/vjti/Desktop/CE_52/RR.py
Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
P1      0               8               9               17
P2      1               4               13              17
P3      2               9               15              24
P4      3               5               15              20

Average Waiting Time: 13.00
Average Turnaround Time: 19.50
vjti@vjti-OptiPlex-3050-AIO:~/Desktop/CE_52$
```

CONCLUSION:- Hence, in this lab, we have learned about various CPU scheduling algorithms, including First Come First Serve, Shortest Job First, Round Robin, and Priority Scheduling. We explored their definitions, advantages, disadvantages, and applications, gaining insights into how these algorithms affect process management and system performance in different computing environments. This understanding is essential for optimizing resource allocation and improving overall system efficiency.