Problem Statement:

Operating systems rely on robust memory management techniques, efficient CPU scheduling policies, and optimized file allocation strategies to manage hardware resources effectively. This lab aims to simulate various such components using Python. Students will implement and analyze Priority and Round Robin scheduling, simulate file allocation techniques (Sequential and Indexed), and explore memory management strategies (MFT, MVT, Worst-fit, Best-fit, First-fit). These implementations will reinforce theoretical OS concepts through hands-on coding experience.

1. Introduction

Operating systems are responsible for efficient resource management including **CPU scheduling, memory allocation, and file storage management**. Understanding these concepts through practical simulations helps bridge the gap between theory and implementation.

This lab focuses on simulating:

1. CPU scheduling algorithms – **Priority Scheduling** and **Round Robin**
2. File allocation strategies – **Sequential** and **Indexed**
3. Memory management – **MFT**, **MVT**, and **Contiguous Allocation (First-fit, Best-fit, Worst-fit)**

Python programming is used to implement these simulations, providing hands-on experience with OS concepts.

---

2. Objectives

1. Implement and analyze **CPU scheduling algorithms** and compute **waiting & turnaround times**.
2. Simulate **sequential and indexed file allocation** strategies.
3. Demonstrate memory management techniques, including **fixed and variable partitions**, and contiguous allocation strategies.
4. Reinforce theoretical OS concepts through **practical coding** and visualization.

---

3. Tools & Technology Used

- **Programming Language:** Python 3.x

- **Python Libraries:** os, time, multiprocessing (built-in)
- **OS:** Linux (recommended, but works on Windows/Mac)
- **IDE:** VS Code / PyCharm / Jupyter Notebook
- **Version Control:** Git & GitHub

---

4. Assignment Tasks and Implementation

Task 1: CPU Scheduling

Algorithms Implemented:

1. **Priority Scheduling** (non-preemptive)
2. **Round Robin Scheduling**

Implementation Details:

- Input: Number of processes, burst times, priorities (for priority), time quantum (for RR)
- Output: Gantt chart representation, Average Waiting Time (AWT), Average Turnaround Time (TAT)
- Logic:
    - Priority: Processes sorted by priority and executed sequentially
    - Round Robin: Each process gets CPU for time quantum until completion

Code Snapshot:

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

Sample Output:

```
Priority Scheduling:
PID   BT   Priority  WT   TAT
1     10   2         0    10
```

```
2     5    1         10   15
Average Waiting Time: 5.0
Average Turnaround Time: 12.5
```

Observation: Priority scheduling favors higher-priority processes; Round Robin ensures fairness but may increase AWT.

Task 2: Sequential File Allocation

Concept: Files occupy contiguous blocks in memory.

Implementation Details:

- Input: Total blocks, starting block, file size
- Checks if requested blocks are free before allocation
- Updates block status

Code Snapshot:

```
total_blocks = int(input("Enter total number of blocks: "))
block_status = [0] * total_blocks
```

Sample Output:

```
File 1 allocated from block 0 to 4
File 2 cannot be allocated (blocks occupied)
```

Observation: Simple to implement but suffers from external fragmentation.

Task 3: Indexed File Allocation

Concept: Each file has an index block pointing to actual data blocks.

Implementation Details:

- `Input: Index block, number of data blocks, list of data block numbers`
- `Ensures index and data blocks are free`
- `Updates block status accordingly`

Sample Output:

```
File 1 allocated with index block 2 -> [5, 6, 7]
```

```
File 2 allocation failed (block 6 already allocated)
```

Observation: Avoids external fragmentation; slightly more complex than sequential allocation.

Task 4: Contiguous Memory Allocation

Strategies Implemented:

1. **First-fit:** Allocates in the first partition that fits
2. **Best-fit:** Allocates in the smallest partition that fits
3. **Worst-fit:** Allocates in the largest partition

Code Snapshot:

```
partitions = list(map(int, input("Enter partition sizes: 
").split()))
```

```
processes = list(map(int, input("Enter process sizes: ").split()))
```

Sample Output (First-fit):

```
Process 1 allocated in Partition 1
```

```
Process 2 allocated in Partition 3
```

```
Process 3 cannot be allocated
```

Observation: Best-fit reduces leftover memory, worst-fit reduces large unused partitions, first-fit is fastest.

Task 5: MFT & MVT Memory Management

MFT (Fixed Partition Table):

- Memory divided into fixed-size partitions
- Processes allocated if they fit

MVT (Variable Partition Table):

- Memory allocated dynamically based on process size
- Remaining memory updated

Sample Output:

```
MFT Simulation:
```

```
Process 1 allocated.
Process 2 too large for fixed partition.
```

MVT Simulation:

Process 1 allocated.

Process 2 allocated.

Process 3 cannot be allocated.

Observation: MVT is more flexible than MFT; fixed partitions may cause internal fragmentation.

---

5. Analysis and Results

Task	**Metric/Observation**
CPU Scheduling	Priority scheduling favors higher priority, RR is fairer
Sequential Allocation	Simple, but external fragmentation may occur
Indexed Allocation	Efficient, avoids fragmentation, supports non-contiguous blocks
Memory Allocation	First/Best/Worst fit balance speed vs memory utilization
MFT & MVT	Fixed partitions may waste memory, variable partitions are flexible

---

6. Conclusion

This lab successfully demonstrates:

- Implementation of CPU scheduling algorithms and memory allocation strategies
- Differences between sequential and indexed file allocation
- Practical understanding of OS concepts such as fragmentation, waiting time, and memory utilization

Through Python simulation, theoretical concepts are reinforced, preparing students for real-world OS problem solving.