

# **Operating Systems Lab Assignment - 3 Report**

**Name:** Gunjan Joshi

**Roll no.** 2301010062

**Course Code:** ENCS351

**Program :** B.Tech CSE

**Experiment: Simulation of File Allocation, Memory Management, and Scheduling in Python**

## **Objective:**

The objective of this lab is to understand and demonstrate Interprocess Communication (IPC) using:

- Pipes (unidirectional communication)
- Shared Memory (fast memory-based communication)
- Synchronization techniques to avoid race conditions
- Parent–Child communication models
- Process creation and data exchange

This experiment helps simulate how operating systems allow multiple processes to exchange data efficiently.

## **Tools Used:**

Python 3.x

Linux Environment (Ubuntu/WSL via Terminal)

os module – for pipe creation and process forking

multiprocessing module – for shared memory and synchronization

logging module

time module

## Tasks Performed:

- Task 1(CPU Scheduling with Gantt Chart):

**Description:** A Python program was written to simulate:

- Priority Scheduling (lower number → higher priority)
- Round Robin Scheduling using fixed time quantum

The program accepts burst time, priority values, and process counts. It calculates:

- Waiting Time (WT)
- Turnaround Time (TAT)
- Average WT and TAT
- Scheduling order as Gantt Chart (printed in sequence)

Outcome:

Correct scheduling order with computed WT and TAT for all processes.

## INPUT:

```
⚡ os3task1.py > ...
1  # Priority Scheduling Simulation
2  processes = []
3  n = int(input("Enter number of processes: "))
4  for i in range(n):
5      bt = int(input(f"Enter Burst Time for P{i+1}: "))
6      pr = int(input(f"Enter Priority (lower number = higher priority) for P{i+1}: "))
7      processes.append((i+1, bt, pr))
8  processes.sort(key=lambda x: x[2])
9  wt = 0
10 total_wt = 0
11 total_tt = 0
12 print("\nPriority Scheduling:")
13 print("PID\tBT\tPriority\tWT\tTAT")
14 for pid, bt, pr in processes:
15     tat = wt + bt
16     print(f"{pid}\t{bt}\t{pr}\t{wt}\t{tat}")
17     total_wt += wt
18     total_tt += tat
19     wt += bt
20 print(f"Average Waiting Time: {total_wt / n}")
21 print(f"Average Turnaround Time: {total_tt / n}")
```

## OUTPUT:

```
● gunjan_linux_2025@LAPTOP-SIB0FPNM:~$ /bin/python3 /home/gunjan_linux_2025/os3task1.py
Enter number of processes: 4
Enter Burst Time for P1: 10
Enter Priority (lower number = higher priority) for P1: 5
Enter Burst Time for P2: 20
Enter Priority (lower number = higher priority) for P2: 6
Enter Burst Time for P3: 30
Enter Priority (lower number = higher priority) for P3: 7
Enter Burst Time for P4: 40
Enter Priority (lower number = higher priority) for P4: 8

Priority Scheduling:
PID      BT      Priority      WT      TAT
1        10      5            0       10
2        20      6            10      30
3        30      7            30      60
4        40      8            60      100
Average Waiting Time: 25.0
Average Turnaround Time: 50.0
```

- **Task 2 (Sequential File Allocation):**

**Description:** A Python script was implemented to simulate block-wise sequential file allocation.

The user provides:

- Total blocks
- Starting block
- File length

The system checks contiguous free space and allocates the file if possible.

Outcome:

Displays allocation success or failure depending on block availability.

## INPUT:

```
⚡ os3task2.py > ...
1  total_blocks = int(input("Enter total number of blocks: "))
2  block_status = [0] * total_blocks
3
4  n = int(input("Enter number of files: "))
5  for i in range(n):
6      start = int(input(f"Enter starting block for file {i+1}: "))
7      length = int(input(f"Enter length of file {i+1}: "))
8      allocated = True
9      for j in range(start, start+length):
10         if j >= total_blocks or block_status[j] == 1:
11             allocated = False
12             break
13         if allocated:
14             for j in range(start, start+length):
15                 block_status[j] = 1
16             print(f"File {i+1} allocated from block {start} to {start+length-1}")
17         else:
18             print(f"File {i+1} cannot be allocated.")
```

## OUTPUT:

```
● gunjan_linux_2025@LAPTOP-SIBOPNM:~$ /bin/python3 /home/gunjan_linux_2025/os3task2.py
Enter total number of blocks: 4
Enter number of files: 3
Enter starting block for file 1: 1
Enter length of file 1: 10
File 1 cannot be allocated.
Enter starting block for file 2: 2
Enter length of file 2: 20
File 2 cannot be allocated.
Enter starting block for file 3: 3
Enter length of file 3: 1
File 3 allocated from block 3 to 3
```

- **Task 3 (Indexed File Allocation):**

**Description:** Simulated indexed allocation by assigning:

- Index block
- List of data blocks

The system verifies block availability and assigns the index → data block mapping.

Outcome:

Correct allocation table showing index block → data blocks.

**INPUT:**

```
❷ os3task3.py > ...
1  total_blocks = int(input("Enter total number of blocks: "))
2  block_status = [0] * total_blocks
3  n = int(input("Enter number of files: "))
4  for i in range(n):
5      index = int(input(f"Enter index block for file {i+1}: "))
6      if block_status[index] == 1:
7          print("Index block already allocated.")
8          continue
9      count = int(input("Enter number of data blocks: "))
10     data_blocks = list(map(int, input("Enter block numbers: ").split()))
11     if any(block_status[blk] == 1 for blk in data_blocks) or len(data_blocks) != count:
12         print("Block(s) already allocated or invalid input.")
13         continue
14     block_status[index] = 1
15     for blk in data_blocks:
16         block_status[blk] = 1
17     print(F"File {i+1} allocated with index block {index} -> {data_blocks}")
```

**OUTPUT:**

```
● gunjan_linux_2025@LAPTOP-SIB0FPNM:~/bin$ /bin/python3 /home/gunjan_linux_2025/os3task3.py
Enter total number of blocks: 20
Enter number of files: 2
Enter index block for file 1: 5
Enter number of data blocks: 3
Enter block numbers: 2 4 6
File 1 allocated with index block 5 -> [2, 4, 6]
Enter index block for file 2: 9
Enter number of data blocks: 2
Enter block numbers: 1 3
File 2 allocated with index block 9 -> [1, 3]
❸ gunjan_linux_2025@LAPTOP-SIB0FPNM:~/bin$
```

- **Task 4 (Contiguous Memory Allocation):**

**Description:** Simulated:

- First-Fit
- Best-Fit
- Worst-Fit

For each strategy, partitions and process sizes were taken as input.

Processes were allocated based on the strategy rules, and remaining partition size was updated.

Outcome:

Displayed which process was allocated to which partition for all strategies.

**INPUT:**

```
⌚ os3task4.py > ...
1  def allocate_memory(strategy):
2      partitions = list(map(int, input("Enter partition sizes: ").split()))
3      processes = list(map(int, input("Enter process sizes: ").split()))
4      allocation = [-1] * len(processes)
5
6      for i, psize in enumerate(processes):
7          idx = -1
8          if strategy == "first":
9              for j, part in enumerate(partitions):
10                  if part >= psize:
11                      idx = j
12                      break
13          elif strategy == "best":
14              best_fit = float("inf")
15              for j, part in enumerate(partitions):
16                  if part >= psize and part < best_fit:
17                      best_fit = part
18                      idx = j
19          elif strategy == "worst":
20              worst_fit = -1
21              for j, part in enumerate(partitions):
22                  if part >= psize and part > worst_fit:
23                      worst_fit = part
24                      idx = j
25          if idx != -1:
26              allocation[i] = idx
27              partitions[idx] -= psize
28
29      for i, a in enumerate(allocation):
30          if a != -1:
31              print(f"Process {i+1} allocated in Partition {a+1}")
32          else:
33              print(f"Process {i+1} cannot be allocated")
34
35  allocate_memory("first")
36  allocate_memory("best")
37  allocate_memory("worst")
```

## OUTPUT:

```
● gunjan_linux_2025@LAPTOP-SIB0FPNM:~$ /bin/python3 /home/gunjan_linux_2025/os3task4.py
Enter partition sizes: 100 500 200 300 600
Enter process sizes: 212 417 112 426
Process 1 allocated in Partition 2
Process 2 allocated in Partition 5
Process 3 allocated in Partition 2
Process 4 cannot be allocated
Enter partition sizes: 100 500
Enter process sizes: 50 400
Process 1 allocated in Partition 1
Process 2 allocated in Partition 2
Enter partition sizes: 50 100 150
Enter process sizes: 50 150 100
Process 1 allocated in Partition 3
Process 2 cannot be allocated
Process 3 allocated in Partition 2
```

- **Task 5 (MFT & MVT Memory Management):**

**Description:** Two memory management schemes were simulated:

### **MFT (Multiprogramming with Fixed Tasks)**

- Total memory is divided into fixed-size partitions
- Only processes smaller than or equal to partition size can be allocated

### **MVT (Multiprogramming with Variable Tasks)**

- memory allocated dynamically
- fragmentation reduces as partitions are variable

Outcome:

Processes were allocated sequentially based on memory availability.

System displayed whether memory was sufficient or not.

## INPUT:

```
⚡ os3task5.py > ...
1  def MFT():
2      mem_size = int(input("Enter total memory size: "))
3      part_size = int(input("Enter partition size: "))
4      n = int(input("Enter number of processes: "))
5      partitions = mem_size // part_size
6      print(f"Memory divided into {partitions} partitions")
7      for i in range(n):
8          psize = int(input(f"Enter size of Process {i+1}: "))
9          if psize <= part_size:
10              print(f"Process {i+1} allocated.")
11          else:
12              print(f"Process {i+1} too large for fixed partition.")
13
14  def MVT():
15      mem_size = int(input("Enter total memory size: "))
16      n = int(input("Enter number of processes: "))
17      for i in range(n):
18          psize = int(input(f"Enter size of Process {i+1}: "))
19          if psize <= mem_size:
20              print(f"Process {i+1} allocated.")
21              mem_size -= psize
22          else:
23              print(f"Process {i+1} cannot be allocated. Not enough memory.")
24
25  print("MFT Simulation:")
26  MFT()
27  print("\nMVT Simulation:")
28  MVT()
```

## OUTPUT:

```
● gunjan_linux_2025@LAPTOP-SIB0FPNM:~$ /bin/python3 /home/gunjan_linux_2025/os3task5.py
MFT Simulation:
Enter total memory size: 100
Enter partition size: 30
Enter number of processes: 2
Memory divided into 3 partitions
Enter size of Process 1: 25
Process 1 allocated.
Enter size of Process 2: 35
Process 2 too large for fixed partition.

MVT Simulation:
Enter total memory size: 100
Enter number of processes: 2
Enter size of Process 1: 25
Process 1 allocated.
Enter size of Process 2: 35
Process 2 allocated.
diamond gunjan_linux_2025@LAPTOP-SIB0FPNM:~$
```

## Outputs:

- Correct simulation of Priority and Round Robin Scheduling
- Accurate Sequential & Indexed File Allocation results
- Proper memory allocation for First-fit, Best-fit, Worst-fit
- Correct MFT & MVT allocation behavior
- Screenshots verifying successful execution

## Learning Outcomes:

- Scheduling: Understood how WT, TAT, and scheduling order depend on burst time, priority, and quantum.
- File Allocation: Learned how file blocks are stored and validated by OS.
- Memory Management: Observed external & internal fragmentation across strategies.
- Fixed vs Variable Partitions: Understood differences between MFT rigid partitioning and MVT flexibility.
- Practical Understanding: Reinforced theory by simulating OS behavior.