LAB ASSIGNMENT-03

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

Task 1: CPU Scheduling with Gantt Chart

Write a Python program to simulate Priority and Round Robin scheduling algorithms. Compute average waiting and turnaround times.

Implementation:

```
# Priority Scheduling Simulation
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 (lower number = higher priority) for P{i+1}: "))
    processes.append((i+1, bt, pr))
processes.sort(key=lambda x: x[2])
wt = 0
total_wt = 0
total_tt = 0
print("\nPriority Scheduling:")
print("PID\tBT\tPriority\tWT\tTAT")
for pid, bt, pr in processes:
    tat = wt + bt
    print(f"{pid}\t{bt}\t{pr}\t\t{wt}\t{tat}")
    total_wt += wt
    total_tt += tat
```

```
    wt += bt
print(f"Average Waiting Time: {total_wt / n}")
print(f"Average Turnaround Time: {total_tt / n}")
```

Output:

```
Enter number of processes: 3
Enter Burst Time for P1: 5
Enter Priority (lower number = higher priority) for P1: 2
Enter Burst Time for P2: 3
Enter Priority (lower number = higher priority) for P2: 1
Enter Burst Time for P3: 8
Enter Priority (lower number = higher priority) for P3: 3

Priority Scheduling:
PID     BT      Priority        WT      TAT
2       3       1               0       3
1       5       2               3       8
3       8       3               8       16
Average Waiting Time: 3.6666666666666665
Average Turnaround Time: 9.0
```

Task 2: Sequential File Allocation

Write a Python program to simulate sequential file allocation strategy.

Implementation:

```
total_blocks = int(input("Enter total number of blocks: "))
block_status = [0] * total_blocks
n = int(input("Enter number of files: "))
for i in range(n):
    start = int(input(f"Enter starting block for file {i+1}: "))
    length = int(input(f"Enter length of file {i+1}: "))
    allocated = True
    for j in range(start, start+length):
        if j >= total_blocks or block_status[j] == 1:
            allocated = False
            break
    if allocated:
        for j in range(start, start+length):
            block_status[j] = 1
        print(f"File {i+1} allocated from block {start} to {start+length-1}")
```

```
    else:
        print(f"File {i+1} cannot be allocated.")
```

Output:

```
Enter total number of blocks: 12
Enter number of files: 3
Enter starting block for file 1: 2
Enter length of file 1: 4
File 1 allocated from block 2 to 5
Enter starting block for file 2: 6
Enter length of file 2: 3
File 2 allocated from block 6 to 8
Enter starting block for file 3: 4
Enter length of file 3: 4
File 3 cannot be allocated.
```

Task 3: Indexed File Allocation

Write a Python program to simulate indexed file allocation strategy.

Implementation:

```
total_blocks = int(input("Enter total number of blocks: "))
block_status = [0] * total_blocks
n = int(input("Enter number of files: "))
for i in range(n):
    index = int(input(f"Enter index block for file {i+1}: "))
    if block_status[index] == 1:
        print("Index block already allocated.")
        continue
    count = int(input("Enter number of data blocks: "))
    data_blocks = list(map(int, input("Enter block numbers: ").split()))
    if any(block_status[blk] == 1 for blk in data_blocks) or len(data_blocks) != count:
        print("Block(s) already allocated or invalid input.")
        continue
    block_status[index] = 1
```

```
    for blk in data_blocks:
        block_status[blk] = 1
    print(f"File {i+1} allocated with index block {index} -> {data_blocks}")
```

Output:

```
Enter total number of blocks: 12
Enter number of files: 2
Enter index block for file 1: 4
Enter number of data blocks: 3
Enter block numbers: 5 6 7
File 1 allocated with index block 4 -> [5, 6, 7]
Enter index block for file 2: 4
Index block already allocated.
```

Task 4: Contiguous Memory Allocation

Simulate Worst-fit, Best-fit, and First-fit memory allocation strategies.

Implementation:

```
def allocate_memory(strategy):
    partitions = list(map(int, input("Enter partition sizes: ").split()))
    processes = list(map(int, input("Enter process sizes: ").split()))
    allocation = [-1] * len(processes)
    for i, psize in enumerate(processes):
        idx = -1
        if strategy == "first":
            for j, part in enumerate(partitions):
                if part >= psize:
                    idx = j
                    break
        elif strategy == "best":
            best_fit = float("inf")
            for j, part in enumerate(partitions):
                if part >= psize and part < best_fit:
                    best_fit = part
```

```
                    idx = j
        elif strategy == "worst":
            worst_fit = -1
            for j, part in enumerate(partitions):
                if part >= psize and part > worst_fit:
                    worst_fit = part
                    idx = j
        if idx != -1:
            allocation[i] = idx
            partitions[idx] -= psize
    for i, a in enumerate(allocation):
        if a != -1:
            print(f"Process {i+1} allocated in Partition {a+1}")
        else:
            print(f"Process {i+1} cannot be allocated")
allocate_memory("first")
allocate_memory("best")
allocate_memory("worst")
```

Output:

```
Enter partition sizes: 100 500 200 300 600
Enter process sizes: 212 417 112 426

---- FIRST FIT ----
Process 1 (212) allocated to Partition 2
Process 2 (417) allocated to Partition 5
Process 3 (112) allocated to Partition 2
Process 4 (426) cannot be allocated
Remaining Partition Sizes: [100, 176, 200, 300, 183]

---- BEST FIT ----
Process 1 (212) allocated to Partition 4
Process 2 (417) allocated to Partition 2
Process 3 (112) allocated to Partition 3
Process 4 (426) allocated to Partition 5
Remaining Partition Sizes: [100, 83, 88, 88, 174]

---- WORST FIT ----
Process 1 (212) allocated to Partition 5
Process 2 (417) allocated to Partition 2
Process 3 (112) allocated to Partition 5
Process 4 (426) cannot be allocated
Remaining Partition Sizes: [100, 83, 200, 300, 276]
```

Task 5: MFT & MVT Memory Management

Implement MFT (fixed partitions) and MVT (variable partitions) strategies in Python.

Implementation:

```
def MFT():
   mem_size = int(input("Enter total memory size: "))
   part_size = int(input("Enter partition size: "))
   n = int(input("Enter number of processes: "))
   partitions = mem_size // part_size
   print(f"Memory divided into {partitions} partitions")
   for i in range(n):
      psize = int(input(f"Enter size of Process {i+1}: "))
      if psize <= part_size:
         print(f"Process {i+1} allocated.")
      else:
         print(f"Process {i+1} too large for fixed partition.")
def MVT():
   mem_size = int(input("Enter total memory size: "))
   n = int(input("Enter number of processes: "))
   for i in range(n):
      psize = int(input(f"Enter size of Process {i+1}: "))
      if psize <= mem_size:
         print(f"Process {i+1} allocated.")
         mem_size -= psize
      else:
         print(f"Process {i+1} cannot be allocated. Not enough memory.")
print("MFT Simulation:")
MFT()
print("\nMVT Simulation:")
MVT()
```

Output:

```
MFT Simulation:
Enter total memory size: 1000
Enter partition size: 300
Enter number of processes: 4
Memory divided into 3 partitions
Enter size of Process 1: 212
Process 1 allocated.
Enter size of Process 2: 417
Process 2 too large for fixed partition.
Enter size of Process 3: 299
Process 3 allocated.
Enter size of Process 4: 150
Process 4 allocated.

MVT Simulation:
Enter total memory size: 1000
Enter number of processes: 3
Enter size of Process 1: 600
Process 1 allocated.
Enter size of Process 2: 500
Process 2 cannot be allocated. Not enough memory.
Enter size of Process 3: 800
Process 3 cannot be allocated. Not enough memory.
```