ASSIGNMENT-3

NAME-ANJALI
ROLL NO.-2301420021
COURSE-BTech CSE(DS)

Task1- Implement priority and round robin scheduling and calculate turn around time and waiting time.

1. Priority Scheduling :
Code:

```
  GNU nano 8.4                                                  task1_priority.py *
n = int(input("n: "))
bt, at = [], []

for i in range(n):
    b, a = map(int, input().split())
    bt.append(b)
    at.append(a)

q = int(input("q: "))

r = bt[:]
t = 0
ct = [0] * n
Q = []

while True:
    for i in range(n):
        if at[i] <= t and r[i] > 0 and i not in Q:
            Q.append(i)

    if not Q:
        if all(x == 0 for x in r):
            break
        t += 1
        continue

    i = Q.pop(0)
    x = min(q, r[i])
    r[i] -= x
    t += x

    for j in range(n):
        if at[j] <= t and r[j] > 0 and j not in Q:
            Q.append(j)

    if r[i] > 0:
        Q.append(i)
    else:
        ct[i] = t

tat = [ct[i] - at[i] for i in range(n)]
wt = [tat[i] - bt[i] for i in range(n)]

print("P\tCT\tTAT\tWT")
for i in range(n):
    print(f"{i+1}\t{ct[i]}\t{tat[i]}\t{wt[i]}")
```

Output:

```
┌──(anjali㉿Anjali)-[~]
└─$ nano task1_priority.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 task1_priority.py
n: 3
6 1
5 2
3 3
q: 2
P       CT      TAT     WT
1       11      10      4
2       15      13      8
3       15      12      9
Avg TAT = 11.67  Avg WT = 7.0
```

2. Round Robin (RR) :

Code:

```
 GNU nano 8.4                                                    task1.py
n = int(input("Enter number of processes: "))

bt = []
at = []
pr = []

for i in range(n):
    b, a, p = map(int, input(f"P{i+1} (burst arrival priority): ").split())
    bt.append(b)
    at.append(a)
    pr.append(p)

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

t = 0

for _ in range(n):
    avail = [i for i in range(n) if not done[i] and at[i] <= t]

    if not avail:
        t = min(at[i] for i in range(n) if not done[i])
        avail = [i for i in range(n) if at[i] <= t and not done[i]]

    i = min(avail, key=lambda x: pr[x])
    t += bt[i]
    ct[i] = t
    done[i] = 1

for i in range(n):
    tat[i] = ct[i] - at[i]
    wt[i] = tat[i] - bt[i]

print("\nP\tBT\tAT\tPR\tCT\tTAT\tWT")
for i in range(n):
    print(f"P{i+1}\t{bt[i]}\t{at[i]}\t{pr[i]}\t{ct[i]}\t{tat[i]}\t{wt[i]}")

print("\nAvg TAT =", round(sum(tat)/n, 2), " Avg WT =", round(sum(wt)/n, 2))
```

Output:

```
┌──(anjali㉿Anjali)-[~]
└─$ nano task1.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 task1.py
Enter number of processes: 3
P1 (burst arrival priority): 5 1 3
P2 (burst arrival priority): 4 2 2
P3 (burst arrival priority): 9 3 4

P       BT      AT      PR      CT      TAT     WT
P1      5       1       3       6       5       0
P2      4       2       2       10      8       4
P3      9       3       4       19      16      7

Avg TAT = 9.67  Avg WT = 3.67
```

Task 2- Simulate worst-fit, best-fit, first-fit memory allocation strategies.

Code:

```
  GNU nano 8.4                                                    task2_memory_allocation.py *
def first_fit(blocks, processes):
    allocation = [-1] * len(processes)
    for i in range(len(processes)):
        for j in range(len(blocks)):
            if blocks[j] >= processes[i]:
                allocation[i] = j
                blocks[j] -= processes[i]
                break
    return allocation

def best_fit(blocks, processes):
    allocation = [-1] * len(processes)
    for i in range(len(processes)):
        best_index = -1
        for j in range(len(blocks)):
            if blocks[j] >= processes[i]:
                if best_index == -1 or blocks[j] < blocks[best_index]:
                    best_index = j
        if best_index != -1:
            allocation[i] = best_index
            blocks[best_index] -= processes[i]
    return allocation

def worst_fit(blocks, processes):
    allocation = [-1] * len(processes)
    for i in range(len(processes)):
        worst_index = -1
        for j in range(len(blocks)):
            if blocks[j] >= processes[i]:
                if worst_index == -1 or blocks[j] > blocks[worst_index]:
                    worst_index = j
        if worst_index != -1:
            allocation[i] = worst_index
            blocks[worst_index] -= processes[i]
    return allocation

def display_result(strategy_name, allocation, processes):
    print(f"\n{strategy_name} Allocation Result:")
    print("{:<12}{:<12}{:<12}".format("Process", "Size", "Block No."))
    for i in range(len(processes)):
        if allocation[i] != -1:
            print("{:<12}{:<12}{:<12}".format(f"P{i+1}", processes[i], allocation[i] + 1))
        else:
            print("{:<12}{:<12}{:<12}".format(f"P{i+1}", processes[i], "Not Allocated"))
```

```
if __name__ == "__main__":
    n_blocks = int(input("Enter number of memory blocks: "))
    blocks = [int(input(f"Block {i+1} size: ")) for i in range(n_blocks)]

    n_processes = int(input("\nEnter number of processes: "))
    processes = [int(input(f"Process {i+1} size: ")) for i in range(n_processes)]

    ff_allocation = first_fit(blocks.copy(), processes)
    bf_allocation = best_fit(blocks.copy(), processes)
    wf_allocation = worst_fit(blocks.copy(), processes)

    display_result("First Fit", ff_allocation, processes)
    display_result("Best Fit", bf_allocation, processes)
    display_result("Worst Fit", wf_allocation, processes)
```

Output:

```
┌──(anjali㉿Anjali)-[~]
└─$ nano task2_memory_allocation.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 task2_memory_allocation.py
Enter number of memory blocks: 5
Block 1 size: 100
Block 2 size: 500
Block 3 size: 300
Block 4 size: 200
Block 5 size: 600

Enter number of processes: 4
Process 1 size: 213
Process 2 size: 420
Process 3 size: 110
Process 4 size: 450

First Fit Allocation Result:
Process     Size        Block No.
P1          213         2
P2          420         5
P3          110         2
P4          450         Not Allocated

Best Fit Allocation Result:
Process     Size        Block No.
P1          213         3
P2          420         2
P3          110         4
P4          450         5

Worst Fit Allocation Result:
Process     Size        Block No.
P1          213         5
P2          420         2
P3          110         5
P4          450         Not Allocated
```

Task 3 - Implement MFT and MVT Memory Management

Code:

```
  GNU nano 8.4                                                  task3_memory_management.py *
def mft(total_memory, partition_size, process_sizes):
    partitions = total_memory // partition_size
    internal_frag = 0
    external_frag = 0
    allocated = 0

    print("\n--- MFT (Multiprogramming with Fixed Tasks) ---")
    print(f"Total Memory: {total_memory} | Partition Size: {partition_size} | Partitions: {partitions}")
    print("\nProcess\tSize\tStatus\t\tFragmentation")

    for i, size in enumerate(process_sizes):
        if allocated < partitions:
            if size > partition_size:
                print(f"P{i+1}\t{size}\tNot Allocated\t(Process too large)")
            else:
                frag = partition_size - size
                internal_frag += frag
                print(f"P{i+1}\t{size}\tAllocated\tInternal Frag = {frag}")
                allocated += 1
        else:
            print(f"P{i+1}\t{size}\tNot Allocated\t(No free partition left)")

    if allocated < len(process_sizes):
        external_frag = total_memory - (partitions * partition_size)

    print(f"\nTotal Internal Fragmentation: {internal_frag}")
    print(f"Total External Fragmentation: {external_frag}\n")


def mvt(total_memory, process_sizes):
    print("\n--- MVT (Multiprogramming with Variable Tasks) ---")
    remaining = total_memory
    allocated = []
    external_frag = 0

    print(f"Total Memory: {total_memory}")
    print("\nProcess\tSize\tStatus\t\tRemaining Memory")
```

```
    for i, size in enumerate(process_sizes):
        if size <= remaining:
            allocated.append(size)
            remaining -= size
            print(f"P{i+1}\t{size}\tAllocated\t{remaining}")
        else:
            print(f"P{i+1}\t{size}\tNot Allocated\t(Insufficient memory)")

    external_frag = remaining
    print(f"\nTotal External Fragmentation: {external_frag}\n")


if __name__ == "__main__":
    print("Memory Management Simulation (MFT and MVT)")
    total_memory = int(input("Enter total memory size: "))

    choice = input("Choose (1) MFT or (2) MVT: ")

    if choice == '1':
        partition_size = int(input("Enter partition size: "))
        n = int(input("Enter number of processes: "))
        process_sizes = [int(input(f"Enter size of P{i+1}: ")) for i in range(n)]
        mft(total_memory, partition_size, process_sizes)

    elif choice == '2':
        n = int(input("Enter number of processes: "))
        process_sizes = [int(input(f"Enter size of P{i+1}: ")) for i in range(n)]
        mvt(total_memory, process_sizes)

    else:
        print("Invalid choice. Please choose 1 for MFT or 2 for MVT.")
```

Output:

```
┌──(anjali㉿Anjali)-[~]
└─$ nano task3_memory_management.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 task3_memory_management.py
Memory Management Simulation (MFT and MVT)
Enter total memory size: 1000
Choose (1) MFT or (2) MVT: 1
Enter partition size: 400
Enter number of processes: 4
Enter size of P1: 210
Enter size of P2: 420
Enter size of P3: 110
Enter size of P4: 350

--- MFT (Multiprogramming with Fixed Tasks) ---
Total Memory: 1000 | Partition Size: 400 | Partitions: 2

Process Size    Status          Fragmentation
P1      210     Allocated       Internal Frag = 190
P2      420     Not Allocated   (Process too large)
P3      110     Allocated       Internal Frag = 290
P4      350     Not Allocated   (No free partition left)

Total Internal Fragmentation: 480
Total External Fragmentation: 200


┌──(anjali㉿Anjali)-[~]
└─$ python3 task3_memory_management.py
Memory Management Simulation (MFT and MVT)
Enter total memory size: 1000
Choose (1) MFT or (2) MVT: 2
Enter number of processes: 4
Enter size of P1: 210
Enter size of P2: 420
Enter size of P3: 110
Enter size of P4: 350

--- MVT (Multiprogramming with Variable Tasks) ---
Total Memory: 1000

Process Size    Status          Remaining Memory
P1      210     Allocated       790
P2      420     Allocated       370
P3      110     Allocated       260
P4      350     Not Allocated   (Insufficient memory)

Total External Fragmentation: 260
```