Lab Experiment Sheet - 03

- **Course Code- ENCS351**
- **Course Name – Operating system**
- **Program Name: B.Tech. CSE- AI ML**

- **Student Name: SURYANSH DHAMA**
- **Roll No. 2301730126**
- **GitHub Repository Link : https://github.com/Suryansh-Dhama/OS-Assignment.git**

Problem Title:

Simulation of File Allocation, Memory Management, and Scheduling in Python

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.

Tools/Technology Used:

- Python 3.x
- Built-in Python libraries: os, multiprocessing, time
- Linux OS (optional for simulation)

Learning Objectives:

1. Simulate and analyze CPU scheduling algorithms.
2. Implement file allocation techniques in Python.
3. Demonstrate memory management strategies including MFT and MVT.

Assignment Tasks:

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.

```
# 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}")
```

code :

Code | Blame | 99 lines (92 loc) · 3.3 KB

```python
from collections import deque

def priority_scheduling(processes):
    # processes: list of tuples (pid, burst, priority)
    procs = sorted(processes, key=lambda x: x[2])  # sort by priority (lower first)
    time = 0
    results = []
    total_wt = 0
    total_tat = 0
    gantt = []
    for pid, bt, pr in procs:
        wt = time
        tat = wt + bt
        results.append((pid, bt, pr, wt, tat))
        total_wt += wt
        total_tat += tat
        gantt.append((pid, time, time + bt))
        time += bt
    n = len(processes)
    avg_wt = total_wt / n
    avg_tat = total_tat / n
    return results, avg_wt, avg_tat, gantt

def round_robin(processes, quantum):
    # processes: list of tuples (pid, burst)
    queue = deque()
    for pid, bt in processes:
        queue.append([pid, bt])
    time = 0
    remaining = {pid: bt for pid, bt in processes}
    completion_time = {}
    gantt = []
    # Keep arrival all at time 0 for simplicity (classical RR).
    while queue:
        pid, rem = queue.popleft()
        exec_time = min(rem, quantum)
        gantt.append((pid, time, time + exec_time))
```

Output :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

harshkumarmishra16@Harshs-MacBook-Air lab sheet 3 % /usr/bin/python3 "/Users/harshkumarmishra16/Documents/lab sheet 3/cpu_scheduling.py"
2       2       2       10      12
3       1       3       12      13
Average WT: 7.33, Average TAT: 11.67
Gantt: |P1:0->10 |P2:10->12 |P3:12->13

Enter time quantum for Round Robin: 2

Round Robin Scheduling
PID     BT      WT      TAT
1       10      3       13
2       2       2       4
3       1       4       5
Average WT: 3.00, Average TAT: 7.33
Gantt: |P1:0->2 |P2:2->4 |P3:4->5 |P1:5->7 |P1:7->9 |P1:9->11 |P1:11->13
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

harshkumarmishra16@Harshs-MacBook-Air lab sheet 3 % /usr/bin/python3 "/Users/harshkumarmishra16/Documents/lab sheet 3/cpu_scheduling.py"
Enter Priority (lower number = higher priority) for P1: 1
Enter Burst Time for P2: 2
Enter Priority (lower number = higher priority) for P2: 2
Enter Burst Time for P3: 1
Enter Priority (lower number = higher priority) for P3: 3

Priority Scheduling (non-preemptive)
PID     BT      PR      WT      TAT
1       10      1       0       10
2       2       2       10      12
3       1       3       12      13
Average WT: 7.33, Average TAT: 11.67
Gantt: |P1:0->10 |P2:10->12 |P3:12->13
```

Task 2: Sequential File Allocation

Write a Python program to simulate sequential file allocation strategy.

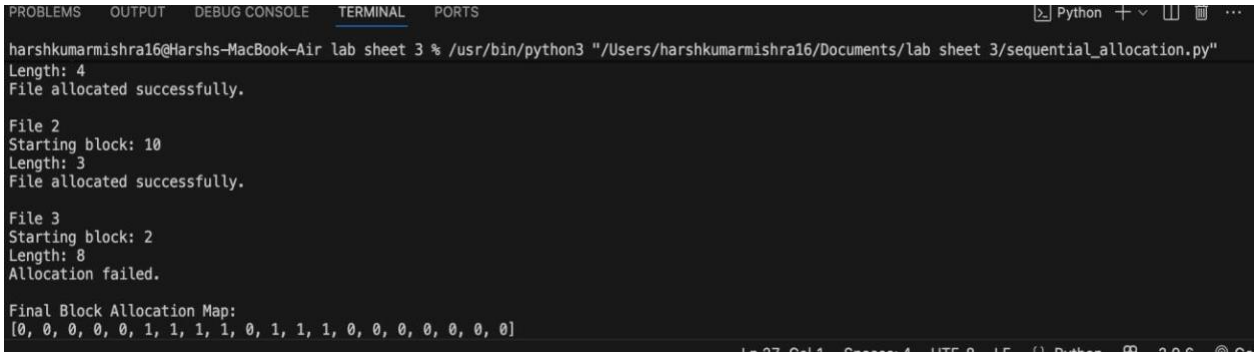
```
total_blocks = int(input("Enter total number of blocks: "))
block_status = [0] * total_blocks

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.")
```

code:

Code | Blame | 38 lines (31 loc) · 1.07 KB

```python
def indexed_allocation():
    total = int(input("Enter total number of blocks: "))
    block_status = [0] * total
    n = int(input("Enter number of files: "))

    for i in range(1, n + 1):
        print(f"\nFile {i}")
        index = int(input("Enter index block: "))
        if block_status[index] == 1:
            print("Index block already used.")
            continue

        count = int(input("Enter number of data blocks: "))
        data = list(map(int, input("Enter data block numbers: ").split()))

        if len(data) != count:
            print("Mismatch in data block count.")
            continue

        flag = False
        for b in data:
            if b >= total or block_status[b] == 1:
                flag = True
                break

        if flag:
            print("Allocation Failed.")
        else:
            block_status[index] = 1
            for b in data:
                block_status[b] = 1
            print("Indexed File Allocated.")

    print("\nFinal Allocation Map:")
    print(block_status)

if __name__ == "__main__":
    indexed_allocation()
```

Output :



Task 3: Indexed File Allocation

Write a Python program to simulate indexed file allocation strategy.

```
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}")
```

code:

Code | Blame | 26 lines (23 loc) · 832 Bytes

```python
def sequential_allocation():
    total_blocks = int(input("Enter total no. of blocks: "))
    block_status = [0] * total_blocks
    n = int(input("Enter number of files: "))

    for i in range(1, n + 1):
        print(f"\nFile {i}")
        start = int(input("Starting block: "))
        length = int(input("Length: "))
        valid = True
        for j in range(start, start + length):
            if j >= total_blocks or block_status[j] == 1:
                valid = False
                break
        if valid:
            for j in range(start, start + length):
                block_status[j] = 1
            print("File allocated successfully.")
        else:
            print("Allocation failed.")

    print("\nFinal Block Allocation Map:")
    print(block_status)

if __name__ == "__main__":
    sequential_allocation()
```

Output :

Task 4: Contiguous Memory Allocation

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

```
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")
```

code:

Code Blame 46 lines (37 loc) · 1.37 KB

```python
def allocate_memory(partitions, processes, strategy):
    parts = partitions[:]
    allocation = [-1] * len(processes)

    for i, p in enumerate(processes):

        idx = -1
        if strategy == "first":
            for j, part in enumerate(parts):
                if part >= p:
                    idx = j
                    break

        elif strategy == "best":
            best_idx = -1
            best_size = None
            for j, part in enumerate(parts):
                if part >= p and (best_size is None or part < best_size):
                    best_size = part
                    best_idx = j
            idx = best_idx

        elif strategy == "worst":
            worst_idx = -1
            worst_size = -1
            for j, part in enumerate(parts):
                if part >= p and part > worst_size:
                    worst_size = part
                    worst_idx = j
            idx = worst_idx

        if idx != -1:
            allocation[i] = idx
            parts[idx] -= p

    return allocation, parts
```

Output :

```
harshkumarmishra16@Harshs-MacBook-Air lab sheet 3 % /usr/bin/python3 "/Users/harshkumarmishra16/Documents/lab sheet 3/contiguous_memory_allocation.py"
Enter partition sizes: 100 500 200 300
Enter process sizes: 212 417 112 426

First Fit Allocation: [1, -1, 1, -1]
Remaining: [100, 176, 200, 300]

Best Fit Allocation: [3, 1, 2, -1]
Remaining: [100, 83, 88, 88]

Worst Fit Allocation: [1, -1, 3, -1]
Remaining: [100, 288, 200, 188]
harshkumarmishra16@Harshs-MacBook-Air lab sheet 3 %
```

Task 5: MFT & MVT Memory Management

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

```
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()

code:

Code | Blame | 42 lines (34 loc) · 1.14 KB

```
def MFT(mem_size, part_size, processes):
    partitions = mem_size // part_size
    result = []
    for i, p in enumerate(processes, start=1):
        ok = p <= part_size
        result.append((i, p, ok))
    return partitions, result


def MVT(mem_size, processes):
    remain = mem_size
    alloc = []
    for i, p in enumerate(processes, start=1):
        if p <= remain:
            alloc.append((i, p, True, remain - p))
            remain -= p
        else:
            alloc.append((i, p, False, remain))
    return alloc, remain


if __name__ == "__main__":
    mem = int(input("Enter memory size for MFT: "))
    ps = int(input("Enter partition size: "))
    n = int(input("Number of processes: "))
    procs = [int(input(f"Process {i+1}: ")) for i in range(n)]

    parts, res = MFT(mem, ps, procs)
    print("\nMFT Results:")
    for r in res:
        print(r)

    mem2 = int(input("\nEnter memory size for MVT: "))
    m = int(input("Number of processes: "))
    pro2 = [int(input(f"Process {i+1}: ")) for i in range(m)]

    alloc, remain = MVT(mem2, pro2)
    print("\nMVT Results:")
    for a in alloc:
```

Output :

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                   Python

harshkumarmishra16@Harshs-MacBook-Air lab sheet 3 % /usr/bin/python3 "/Users/harshkumarmishra16/Documents/lab sheet 3/mft_mvt_simulation.py"

Enter memory size for MVT: 1000
Number of processes: 4
Process 1: 212
Process 2: 417
Process 3: 112
Process 4: 426

MVT Results:
(1, 212, True, 788)
(2, 417, True, 371)
(3, 112, True, 259)
(4, 426, False, 259)
Remaining: 259
harshkumarmishra16@Harshs-MacBook-Air lab sheet 3 %
```