

Lab Experiment Sheet - 03

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

- Student Name: AKSHITA GOEL
- Roll No. 2301730231
- GitHub Repository Link : <https://github.com/akshitagoel11/OS-Lab>



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
5 from collections import deque
6
7 def priority_scheduling(processes):
8     # processes: list of tuples (pid, burst, priority)
9     procs = sorted(processes, key=lambda x: x[2]) # sort by priority (lower first)
10    time = 0
11    results = []
12    total_wt = 0
13    total_tat = 0
14    gantt = []
15    for pid, bt, pr in procs:
16        wt = time
17        tat = wt + bt
18        results.append((pid, bt, pr, wt, tat))
19        total_wt += wt
20        total_tat += tat
21        gantt.append((pid, time, time + bt))
22        time += bt
23    n = len(processes)
24    avg_wt = total_wt / n
25    avg_tat = total_tat / n
26    return results, avg_wt, avg_tat, gantt
27
28 def round_robin(processes, quantum):
29     # processes: list of tuples (pid, burst)
30     queue = deque()
31     for pid, bt in processes:
32         queue.append((pid, bt))
33     time = 0
34     remaining = {pid: bt for pid, bt in processes}
35     completion_time = {}
36     gantt = []
37     # Keep arrival all at time 0 for simplicity (classical RR).
38     while queue:
39         pid, rem = queue.popleft()
40         exec_time = min(rem, quantum)
41         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

1  ✓ def indexed_allocation():
2      total = int(input("Enter total number of blocks: "))
3      block_status = [0] * total
4      n = int(input("Enter number of files: "))
5
6      for i in range(1, n + 1):
7          print(f"\nFile {i}")
8          index = int(input("Enter index block: "))
9          if block_status[index] == 1:
10             print("Index block already used.")
11             continue
12
13             count = int(input("Enter number of data blocks: "))
14             data = list(map(int, input("Enter data block numbers: ").split()))
15
16             if len(data) != count:
17                 print("Mismatch in data block count.")
18                 continue
19
20             flag = False
21             for b in data:
22                 if b >= total or block_status[b] == 1:
23                     flag = True
24                     break
25
26             if flag:
27                 print("Allocation Failed.")
28             else:
29                 block_status[index] = 1
30                 for b in data:
31                     block_status[b] = 1
32                 print("Indexed File Allocated.")
33
34             print("\nFinal Allocation Map:")
35             print(block_status)
36
37 if __name__ == "__main__":
38     indexed_allocation()
```

Output :

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
harshkumarmishra16@Harshs-MacBook-Air lab sheet 3 % /usr/bin/python3 "/Users/harshkumarmishra16/Documents/lab sheet 3/sequential_allocation.py"
Length: 4
File allocated successfully.

File 2
Starting block: 10
Length: 3
File allocated successfully.

File 3
Starting block: 2
Length: 8
Allocation failed.

Final Block Allocation Map:
[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]
```

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

```
1  ✓ def sequential_allocation():
2      total_blocks = int(input("Enter total no. of blocks: "))
3      block_status = [0] * total_blocks
4      n = int(input("Enter number of files: "))
5
6      for i in range(1, n + 1):
7          print(f"\nFile {i}")
8          start = int(input("Starting block: "))
9          length = int(input("Length: "))
10         valid = True
11         for j in range(start, start + length):
12             if j >= total_blocks or block_status[j] == 1:
13                 valid = False
14                 break
15         if valid:
16             for j in range(start, start + length):
17                 block_status[j] = 1
18                 print("File allocated successfully.")
19         else:
20             print("Allocation failed.")
21
22         print("\nFinal Block Allocation Map:")
23         print(block_status)
24
25     if __name__ == "__main__":
26         sequential_allocation()
```

Output :


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

File 2
Enter index block: 10
Enter number of data blocks: 2
Enter data block numbers: 11 12
Indexed File Allocated.

File 3
Enter index block: 15
Enter number of data blocks: 4
Enter data block numbers: 16 17 18 19
Indexed File Allocated.

Final Allocation Map:
[0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1]
```

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

```
1  def allocate_memory(partitions, processes, strategy):
2      parts = partitions[:]
3      allocation = [-1] * len(processes)
4
5      for i, p in enumerate(processes):
6
7          idx = -1
8          if strategy == "first":
9              for j, part in enumerate(parts):
10                 if part >= p:
11                     idx = j
12                     break
13
14             elif strategy == "best":
15                 best_idx = -1
16                 best_size = None
17                 for j, part in enumerate(parts):
18                     if part >= p and (best_size is None or part < best_size):
19                         best_size = part
20                         best_idx = j
21                 idx = best_idx
22
23             elif strategy == "worst":
24                 worst_idx = -1
25                 worst_size = -1
26                 for j, part in enumerate(parts):
27                     if part >= p and part > worst_size:
28                         worst_size = part
29                         worst_idx = j
30                 idx = worst_idx
31
32             if idx != -1:
33                 allocation[i] = idx
34                 parts[idx] -= p
35
36     return allocation, parts
37
```

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

1  ✓ def MFT(mem_size, part_size, processes):
2      partitions = mem_size // part_size
3      result = []
4      for i, p in enumerate(processes, start=1):
5          ok = p <= part_size
6          result.append((i, p, ok))
7      return partitions, result
8
9
10 ✓ def MVT(mem_size, processes):
11     remain = mem_size
12     alloc = []
13     for i, p in enumerate(processes, start=1):
14         if p <= remain:
15             alloc.append((i, p, True, remain - p))
16             remain -= p
17         else:
18             alloc.append((i, p, False, remain))
19     return alloc, remain
20
21
22 if __name__ == "__main__":
23     mem = int(input("Enter memory size for MFT: "))
24     ps = int(input("Enter partition size: "))
25     n = int(input("Number of processes: "))
26     procs = [int(input(f"Process {i+1}: ")) for i in range(n)]
27
28     parts, res = MFT(mem, ps, procs)
29     print("\nMFT Results:")
30     for r in res:
31         print(r)
32
33     mem2 = int(input("\nEnter memory size for MVT: "))
34     m = int(input("Number of processes: "))
35     pro2 = [int(input(f"Process {i+1}: ")) for i in range(m)]
36
37     alloc, remain = MVT(mem2, pro2)
38     print("\nMVT Results:")
39     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 %
```