

KR MANGALAM UNIVERSITY

SOHNA ROAD, GURUGRAM



OPERATING SYSTEM LAB FILE

COURSE CODE-ENCS351

Submitted by

Name: Aldrin Debbarma

No.: 2401011482

Program: BTech CSE

Section: 6/F

:Session: 2025/26

Date: 02/12/2025

Submitted To

Dr Satinder Pal Singh Roll

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{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}")
```


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.")
```

AgentsEditor

python

main.py X

main.py

PYTHON

main.py

process_log.txt

README.md

service_dependency...

system_boot_log.txt

system_shutdown_...

```
1 def indexed_file_allocation():
2     print("\n=== INDEXED FILE ALLOCATION SIMULATION ===\n")
3
4     total_blocks = 15          # total disk blocks
5     block_status = [0] * total_blocks  # 0 = free, 1 = allocated
6
7     files = [
8         {"index": 2, "data": [5, 6, 7]},
9         {"index": 4, "data": [8, 9]},
10        {"index": 10, "data": [11, 12, 13]},
11    ]
12
13    for i, file in enumerate(files, start=1):
14        index = file["index"]
15        data_blocks = file["data"]
16
17        print(f"\nFile {i}: Index Block = {index}, Data Blocks = {data_blocks}")
18
19        # Check index block free or not
20        if block_status[index] == 1:
21            print(f"X Index block already allocated. File cannot be allocated.")
22            continue
23
24        # Check data blocks free or not
25        failed = False
```

ProblemsOutputDebug ConsoleTerminalPorts

zsh

```
=== INDEXED FILE ALLOCATION SIMULATION ===

File 1: Index Block = 2, Data Blocks = [5, 6, 7]
✓ File 1 allocated successfully:
  Index Block 2 - Data Blocks [5, 6, 7]

File 2: Index Block = 4, Data Blocks = [8, 9]
✓ File 2 allocated successfully:
  Index Block 4 - Data Blocks [8, 9]

File 3: Index Block = 10, Data Blocks = [11, 12, 13]
✓ File 3 allocated successfully:
  Index Block 10 - Data Blocks [11, 12, 13]

Final Block Allocation Status:
Block 0: Free
Block 1: Free
Block 2: Allocated
Block 3: Free
Block 4: Allocated
Block 5: Allocated
Block 6: Allocated
Block 7: Allocated
Block 8: Allocated
Block 9: Allocated
Block 10: Allocated
Block 11: Allocated
Block 12: Allocated
Block 13: Allocated
Block 14: Free
```

OUTLINE

TIMELINE

WX to generate command

Cursor TabpythonLn 49, Col 26Spaces: 4UTF-8LFPython

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}")
```

AgentsEditor

python

main.py X

PYTHON

main.py

process_log.txt

README.md

service_dependency...

system_boot_log.txt

system_shutdown_...

1

def allocate_memory(strategy, partitions, processes):

42

for i, a in enumerate(allocation):

43

if a != -1:

45

else:

46

print(f"Process {i+1} ({processes[i]} KB) - Not Allocated")

47

print("\nRemaining Partition Sizes:", partitions)

48

print("Original Partition Sizes:", original_partitions)

50

print("=====")

51

52

53

===== MAIN SIMULATION =====

54

if __name__ == "__main__":

55

Example Inputs (Assignment-friendly fixed test)

56

partitions = [100, 500, 200, 300, 600] # memory blocks

57

processes = [212, 417, 112, 426] # process sizes

58

59

allocate_memory("first", partitions.copy(), processes)

60

allocate_memory("best", partitions.copy(), processes)

61

allocate_memory("worst", partitions.copy(), processes)

ProblemsOutputDebug ConsoleTerminalPorts

zsh + v

ashishyadvanshi@Ashishs-MacBook-Air python % python3 main.py

===== FIRST FIT ALLOCATION =====

Process 1 (212 KB) - Partition 2

Process 2 (417 KB) - Partition 5

Process 3 (112 KB) - Partition 2

Process 4 (426 KB) - Not Allocated

Remaining Partition Sizes: [100, 176, 200, 300, 183]

Original Partition Sizes: [100, 500, 200, 300, 600]

=====

===== BEST FIT ALLOCATION =====

Process 1 (212 KB) - Partition 4

Process 2 (417 KB) - Partition 2

Process 3 (112 KB) - Partition 3

Process 4 (426 KB) - Partition 5

Remaining Partition Sizes: [100, 83, 88, 88, 174]

Original Partition Sizes: [100, 500, 200, 300, 600]

=====

===== WORST FIT ALLOCATION =====

Process 1 (212 KB) - Partition 5

Process 2 (417 KB) - Partition 2

Process 3 (112 KB) - Partition 5

Process 4 (426 KB) - Not Allocated

Remaining Partition Sizes: [100, 83, 200, 300, 276]

Original Partition Sizes: [100, 500, 200, 300, 600]

=====

ashishyadvanshi@Ashishs-MacBook-Air python %

OUTLINE

TIMELINE

Cursor Tab

Ln 54, Col 24

Spaces: 4

UTF-8

LF

Python

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")
```

AgentsEditor

python

main.py

PYTHON

main.py

process_log.txt

README.md

service_dependency_...

system_boot_log.txt

system_shutdown_l...

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

0 = free block

1 = occupied block

def sequential_allocation():

print("==== Sequential File Allocation ====\n")

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):

print(f"\n--- File {i+1} ---")

start = int(input("Enter starting block: "))

length = int(input("Enter file length: "))

allocated = True

Check if all blocks from start to start+length are free

for j in range(start, start + length):

if j >= total_blocks or block_status[j] == 1:

allocated = False

break

if allocated:

ProblemsOutputDebug ConsoleTerminalPorts

ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py

==== PRIORITY SCHEDULING ====

PID	BT	Priority	WT	TAT
2	5	1	0	5
3	8	2	5	13
1	10	3	13	23

Gantt Chart:

| P2 | P3 | P1 |

Average Waiting Time = 6.00

Average Turnaround Time = 13.67

==== ROUND ROBIN SCHEDULING ====

PID	BT	WT	TAT
1	10	13	23
2	5	9	14
3	8	14	22

Gantt Chart:

| P1 | P2 | P3 | P1 | P2 | P3 | P1 | P3 | P1 |

Average Waiting Time = 12.00

Average Turnaround Time = 19.67

ashishyaduvanshi@Ashishs-MacBook-Air python %

OUTLINE

TIMELINE

0

Cursor TabLn 37, Col 24Spaces: 4UTF-8LFPython

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()
```

```
def c_look(requests, head):
    print("\n=== C-LOOK DISK SCHEDULING ===\n")
    print(f"Initial Head Position: {head}")
    print(f"Requests: {requests}\n")

    left = [r for r in requests if r < head]
    right = [r for r in requests if r >= head]

    left.sort()
    right.sort()

    order = []
    total_movement = 0
    current = head

    # First go to the right side
    for r in right:
        total_movement += abs(current - r)
        current = r
        order.append(r)

    # Jump to lowest left request
    if left:
        total_movement += abs(current - left[0])
        current = left[0]
```

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py

=== C-LOOK DISK SCHEDULING ===

Initial Head Position: 50
Requests: [95, 180, 34, 119, 11, 123, 62, 64]

Order of Service: [62, 64, 95, 119, 123, 180, 11, 34]
Total Head Movement = 322 cylinders

ashishyaduvanshi@Ashishs-MacBook-Air python %
```

