

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}\t")

    total_wt += wt

    total_tt += tat

    wt += bt

print(f"Average Waiting Time: {total_wt / n}")

print(f"Average Turnaround Time: {total_tt / n}")
```

Agents Editor

main.py X

PYTHON

main.py process_log.txt README.md service_dependency system_boot_log.txt system_shutdown_log.txt

```

1 def priority_scheduling(processes):
2     total_tt = 0
3     total_wt = 0
4     gantt = []
5
6     for pid, bt, pr in processes:
7         wt = 0
8         print(f"({pid})|({bt})|({pr})|{total_wt}|{total_tt}")
9         gantt.append((pid, bt))
10
11         total_wt += wt
12         total_tt += wt
13         wt += bt
14
15     print("\nGantt Chart:")
16     for pid, bt in gantt:
17         print(f" {pid} ", end="")
18     print("\n")
19
20     print("Average Waiting Time = (total_wt/len(processes)),2f")
21     print("Average Turnaround Time = (total_tt/len(processes)),2f")
22
23
24 # ----- ROUND ROBIN SCHEDULING -----
25
26 def round_robin(processes, quantum):
27     print("\n----- ROUND ROBIN SCHEDULING -----")
28
29     n = len(processes)
30     bt = [p[1] for p in processes] # burst times
31     rt = bt[:] # remaining time
32     wt = [0] * n
33     tat = [0] * n
34
35     time_elapsed = 0
36     gantt = []
37
38     while sum(rt) > 0:
39         for i in range(n):
40             if rt[i] >= quantum:
41                 rt[i] -= quantum
42                 tat[i] += quantum
43                 time_elapsed += quantum
44                 if rt[i] == 0:
45                     gantt.append((i+1, bt[i]))
46             else:
47                 rt[i] = 0
48                 tat[i] += rt[i]
49                 time_elapsed += rt[i]
50                 gantt.append((i+1, bt[i]))
51
52     print("\nGantt Chart:")
53     for i in range(n):
54         print(f" {i+1} ", end="")
55     print("\n")
56
57     print("Average Waiting Time = 6.00")
58     print("Average Turnaround Time = 13.67")
59
60     print("\n----- ROUND ROBIN SCHEDULING -----")
61
62     n = len(processes)
63     bt = [p[1] for p in processes] # burst times
64     rt = bt[:] # remaining time
65     wt = [0] * n
66     tat = [0] * n
67
68     time_elapsed = 0
69     gantt = []
70
71     while sum(rt) > 0:
72         for i in range(n):
73             if rt[i] >= quantum:
74                 rt[i] -= quantum
75                 tat[i] += quantum
76                 time_elapsed += quantum
77                 if rt[i] == 0:
78                     gantt.append((i+1, bt[i]))
79             else:
80                 rt[i] = 0
81                 tat[i] += rt[i]
82                 time_elapsed += rt[i]
83                 gantt.append((i+1, bt[i]))
84
85     print("\nGantt Chart:")
86     for i in range(n):
87         print(f" {i+1} ", end="")
88     print("\n")
89
90     print("Average Waiting Time = 12.00")
91     print("Average Turnaround Time = 19.67")
92
93 ashishyadavanshi@ashish-MacBook-Air ~ % python3 main.py
94
95 ----- PRIORITY SCHEDULING -----
96
97 PID BT Priority WT TAT
98 2 5 1 0 5
99 3 8 2 5 13
100 1 10 3 13 23
101
102 Gantt Chart:
103 | P2 | P3 | P1 |
104
105 Average Waiting Time = 6.00
106 Average Turnaround Time = 13.67
107
108 ----- ROUND ROBIN SCHEDULING -----
109
110 PID BT WT TAT
111 1 10 13 23
112 2 5 9 14
113 3 8 14 22
114
115 Gantt Chart:
116 | P1 | P2 | P3 | P1 | P2 | P3 | P1 | P3 | P1 |
117
118 Average Waiting Time = 12.00
119 Average Turnaround Time = 19.67
120
121 ashishyadavanshi@ashish-MacBook-Air ~ %

```

OUTLINE

TIMELINE

0 0 △ 0

JK to generate command

Cursor Tab Ln 81, Col 24 Spaces: 4 UTF-8 LF Python

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.")
```

The screenshot shows a Python code editor interface with the following details:

- Editor Tab:** The tab bar at the top shows "Agents" and "Editor".
- Search Bar:** A search bar in the top right corner contains the text "python".
- Sidebar:** On the left, there's a sidebar titled "PYTHON" containing files: "main.py", "process_log.txt", "README.md", "service_dependency...", "system_boot_log.txt", and "system_shutdown_...".
- Main Editor Area:** The main area displays the content of "main.py". The code performs indexed file allocation simulation, checking index and data block statuses.
- Output Area:** Below the editor, the terminal output shows the execution of the script:
 - Output starts with "==== INDEXED FILE ALLOCATION SIMULATION ====".
 - File 1 allocation status: "File 1 allocated successfully: Index Block 2 - Data Blocks [5, 6, 7]".
 - File 2 allocation status: "File 2 allocated successfully: Index Block 4 - Data Blocks [8, 9]".
 - File 3 allocation status: "File 3 allocated successfully: Index Block 10 - Data Blocks [11, 12, 13]".
 - Final Block Allocation Status: Shows a list of blocks from 0 to 14, where blocks 0, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13 are free, and blocks 2, 4, 14 are allocated.
- Bottom Navigation:** The bottom navigation bar includes tabs for "Problems", "Output", "Debug Console", "Terminal", and "Ports".
- Bottom Status:** The bottom right corner shows the command line prompt "zsh" and other system information like "Ln 49, Col 26" and "UTF-8 LF Python".

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}')
```

The screenshot shows a code editor interface with a dark theme. The top bar includes tabs for 'Agents' and 'Editor', a search bar containing 'Q python', and various icons for file operations. The left sidebar lists project files: 'main.py', 'process_log.txt', 'README.md', 'service_dependency...', 'system_boot_log.txt', and 'system_shutdown_...'. The main editor area displays the 'main.py' file content:

```
def allocate_memory(strategy, partitions, processes):
    for i, a in enumerate(allocation):
        if a != -1:
            else:
                print("Process {i+1} ({processes[i]} KB) - Not Allocated")
        print("\nRemaining Partition Sizes:", partitions)
        print("Original Partition Sizes:", original_partitions)
        print("=====")

# ----- MAIN SIMULATION -----
if __name__ == "__main__":
    # Example inputs (Assignment-friendly fixed test)
    partitions = [100, 500, 200, 300, 600] # memory blocks
    processes = [212, 417, 112, 426] # process sizes

    allocate_memory("first", partitions.copy(), processes)
    allocate_memory("best", partitions.copy(), processes)
    allocate_memory("worst", partitions.copy(), processes)
```

The bottom half of the screen shows the terminal output of running the script:

```
ashishyaduvanshi@ashish-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]
```

The terminal also shows the prompt 'ashishyaduvanshi@ashish-MacBook-Air %' and the status bar at the bottom indicates 'Cursor Tab' and file statistics like 'Ln 54, Col 24'.

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 >= psizes:

                    idx = j

                    break

        elif strategy == "best":

            best_fit = float("inf")

            for j, part in enumerate(partitions):

                if part >= psizes and part < best_fit:

                    best_fit = part

                    idx = j

        elif strategy == "worst":

            worst_fit = -1

            for j, part in enumerate(partitions):

                if part >= psizes 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")
```

Agents Editor ⌂ python

main.py

```
1 # 0 = free block
2 # 1 = occupied block
3
4 def sequential_allocation():
5     print("==== Sequential File Allocation ====\n")
6
7     total_blocks = int(input("Enter total number of blocks: "))
8     block_status = [0] * total_blocks
9
10    n = int(input("Enter number of files: "))
11
12    for i in range(n):
13        print("\n---- File {i+1} ---")
14        start = int(input("Enter starting block: "))
15        length = int(input("Enter file length: "))
16
17        allocated = True
18
19        # check if all blocks from start to start+length are free
20        for j in range(start, start + length):
21            if j >= total_blocks or block_status[j] == 1:
22                allocated = False
23                break
24
25    if allocated:
```

Problems Output Debug Console Terminal Ports

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

KK to generate command

Cursor Tab Q Ln 37, Col 24 Spaces: 4 UTF-8 LF Python

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:

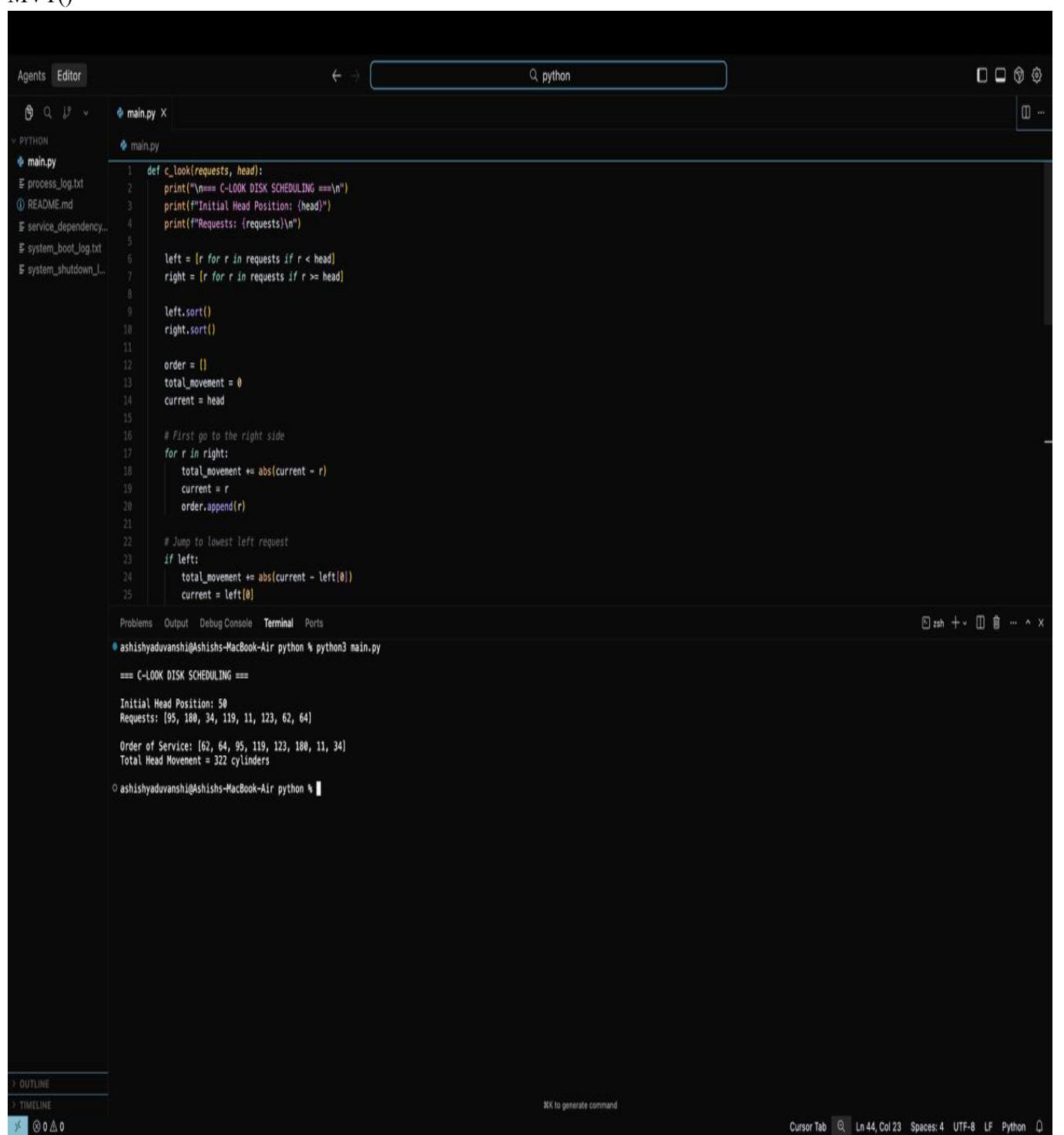
            print(f"Process {i+1} allocated.")

            mem_size -= psize

        else:

            print(f"Process {i+1} cannot be allocated. Not enough memory.")
```

```
print("MFT Simulation:")  
MFT()  
  
print("\nMVT Simulation:")  
MVT()
```



The screenshot shows a code editor interface with a dark theme. The top bar includes tabs for 'Agents' and 'Editor', a search bar containing 'python', and various window control icons. The main area displays a Python file named 'main.py'. The code implements the C-Look disk scheduling algorithm. It starts by printing initial head position and requests, then sorts them into left and right lists based on their distance from the current head position. It then iterates through the right list, moving the head to each request and appending it to the 'order' list. After processing the right list, it checks if there are any requests in the left list. If so, it moves the head to the lowest-left request and appends it to the 'order' list. Finally, it prints the total head movement. The terminal below the editor shows the execution of the script and its output.

```
Agents Editor ← → Q python ...  
Agents Editor ← → Q python ...  
main.py  
main.py  
process_log.txt  
README.md  
service_dependency...  
system_boot_log.txt  
system_shutdown_...  
main.py  
1 def c_look(requests, head):  
2     print("==== C-LOOK DISK SCHEDULING ===\n")  
3     print("Initial Head Position: (head)")  
4     print("Requests: {requests}\n")  
5  
6     left = [r for r in requests if r < head]  
7     right = [r for r in requests if r >= head]  
8  
9     left.sort()  
10    right.sort()  
11  
12    order = []  
13    total_movement = 0  
14    current = head  
15  
16    # First go to the right side  
17    for r in right:  
18        total_movement += abs(current - r)  
19        current = r  
20        order.append(r)  
21  
22    # Jump to lowest left request  
23    if left:  
24        total_movement += abs(current - left[0])  
25        current = left[0]  
26  
27    print("Order of Service: {order}")  
28    print(f"Total Head Movement = {total_movement} cylinders")  
29  
30  
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 %
```

OUTLINE
TIMELINE

⌘K to generate command

Cursor Tab ⌘Q Ln 44, Col 23 Spaces: 4 UTF-8 LF Python ⌘

