Question 1

To solve this problem, we need to allocate memory to processes using different partition allocation schemes. Let's examine First Fit, Best Fit, and Worst Fit memory allocation schemes.

### Given:

- Memory blocks: 150K, 350K

- Process requests: 300K, 25K, 125K, 50K

### 1. First Fit Allocation

This scheme allocates the first available block that is large enough to accommodate the process.

- \*\*Process 300K:\*\* The first block (150K) is too small. It goes to the next available block (350K), which can accommodate it. After allocation, 350K becomes 50K.

- \*\*Process 25K:\*\* Allocated to the first block (150K), leaving 125K.

- \*\*Process 125K:\*\* Fits into the remaining space of 125K in the first block.

- \*\*Process 50K:\*\* The remaining space of 50K from the 350K block is exactly enough to allocate 50K.

\*\*Result:\*\*

- Memory Block 1 (150K): Allocated to 25K and 125K.

- Memory Block 2 (350K): Allocated to 300K and 50K.

### 2. Best Fit Allocation

This scheme allocates the smallest available block that is large enough for the process, aiming to leave the least unused space.

- \*\*Process 300K:\*\* Allocates to the larger block (350K). Leaves 50K.

- \*\*Process 25K:\*\* Allocates to 50K remaining in the larger block.

- \*\*Process 125K:\*\* Allocates to 150K, leaving 25K.

- \*\*Process 50K:\*\* Allocates to the remaining 25K from Block 1.

### 3. Worst Fit Allocation

This scheme allocates to the largest available block, aiming to leave the largest remaining block.

Let's continue with the Worst Fit allocation, where the process is allocated to the largest available block that can accommodate it.

### 3. Worst Fit Allocation

We aim to allocate each process to the largest available memory block.

- \*\*Process 300K:\*\* The largest block available is 350K, so we allocate the 300K process to this block. After this allocation, 50K remains in the 350K block.

- \*\*Process 25K:\*\* The largest available block now is the 150K block. We allocate the 25K process to this block, leaving 125K.

- \*\*Process 125K:\*\* The largest block available is the remaining 125K in the 150K block. We allocate the 125K process here, leaving 0K.

- \*\*Process 50K:\*\* The only remaining space is the 50K leftover in the 350K block, which is a perfect fit for the 50K process.

\*\*Final Allocation:\*\*

- \*\*Memory Block 1 (150K):\*\* Allocated to 25K and 125K (now fully utilized).

- \*\*Memory Block 2 (350K):\*\* Allocated to 300K and 50K (now fully utilized).

### Summary of Results:

#### 1. First Fit:

- Block 1 (150K): 25K + 125K

- Block 2 (350K): 300K + 50K

#### 2. Best Fit:

- Block 1 (150K): 125K + 25K (leftover)

- Block 2 (350K): 300K + 50K

#### 3. Worst Fit:

Bestallocated

Question 2

Let me solve this step by step.

1) First, let's clarify the scenario:

- Disk has cylinders 0 to 50 (51 cylinders total)

- Starting position: 11

- Request queue in order: 1, 36, 16, 34, 9, 12

2) Let's calculate for each algorithm:

FCFS (First Come First Serve):

- Path: 11 → 1 → 36 → 16 → 34 → 9 → 12

- Distance: |11-1| + |1-36| + |36-16| + |16-34| + |34-9| + |9-12|

- = 10 + 35 + 20 + 18 + 25 + 3

- Total = 111 cylinders

SSTF (Shortest Seek Time First):

- At 11, closest is: 12 (distance 1)

- At 12, closest is: 9 (distance 3)

- At 9, closest is: 1 (distance 8)

- At 1, closest is: 16 (distance 15)

- At 16, closest is: 34 (distance 18)

- At 34, closest is: 36 (distance 2)

- Path: 11 → 12 → 9 → 1 → 16 → 34 → 36

- Total = 1 + 3 + 8 + 15 + 18 + 2 = 47 cylinders

SCAN (moves until end even if no requests):

- Starting at 11, going towards 50

- Path: 11 → 12 → 16 → 34 → 36 → 50 → 9 → 1

- Distance: |11-50| + |50-1|

- = 39 + 49

- Total = 88 cylinders

LOOK (like SCAN but reverses at last request):

- Starting at 11, going towards 50

- Path: 11 → 12 → 16 → 34 → 36 → 9 → 1

- Distance: |11-36| + |36-1|

- = 25 + 35

- Total = 60 cylinders

Therefore:

FCFS: 111 cylinders

SSTF: 47 cylinders

SCAN: 88 cylinders

LOOK: 60 cylinders

The most efficient algorithm in this case is SSTF with 47 cylinders of total movement.

Question 3

Let's calculate the number of page faults and page hits for the given reference string using the FIFO and LRU page replacement algorithms.

### Given:

- Reference string: 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0

- Memory with 3 frames

### i. FIFO (First In, First Out) Page Replacement Algorithm

\*\*Steps:\*\*

1. \*\*6:\*\* Page fault (Frames: [6, -, -])

2. \*\*1:\*\* Page fault (Frames: [6, 1, -])

3. \*\*1:\*\* Page hit (Frames: [6, 1, -])

4. \*\*2:\*\* Page fault (Frames: [6, 1, 2])

5. \*\*0:\*\* Page fault (Replace 6 → Frames: [0, 1, 2])

6. \*\*3:\*\* Page fault (Replace 1 → Frames: [0, 3, 2])

7. \*\*4:\*\* Page fault (Replace 2 → Frames: [0, 3, 4])

8. \*\*6:\*\* Page fault (Replace 0 → Frames: [6, 3, 4])

9. \*\*0:\*\* Page fault (Replace 3 → Frames: [6, 0, 4])

10. \*\*2:\*\* Page fault (Replace 4 → Frames: [6, 0, 2])

11. \*\*1:\*\* Page fault (Replace 6 → Frames: [1, 0, 2])

12. \*\*2:\*\* Page hit (Frames: [1, 0, 2])

13. \*\*1:\*\* Page hit (Frames: [1, 0, 2])

14. \*\*2:\*\* Page hit (Frames: [1, 0, 2])

15. \*\*0:\*\* Page hit (Frames: [1, 0, 2])

16. \*\*3:\*\* Page fault (Replace 1 → Frames: [3, 0, 2])

17. \*\*2:\*\* Page hit (Frames: [3, 0, 2])

18. \*\*1:\*\* Page fault (Replace 0 → Frames: [3, 1, 2])

19. \*\*2:\*\* Page hit (Frames: [3, 1, 2])

20. \*\*0:\*\* Page fault (Replace 2 → Frames: [3, 1, 0])

\*\*Total Page Faults:\*\* 15

\*\*Total Page Hits:\*\* 5

### ii. LRU (Least Recently Used) Page Replacement Algorithm

\*\*Steps:\*\*

1. \*\*6:\*\* Page fault (Frames: [6, -, -])

2. \*\*1:\*\* Page fault (Frames: [6, 1, -])

3. \*\*1:\*\* Page hit (Frames: [6, 1, -])

4. \*\*2:\*\* Page fault (Frames: [6, 1, 2])

5. \*\*0:\*\* Page fault (Replace 6 → Frames: [0, 1, 2])

6. \*\*3:\*\* Page fault (Replace 1 → Frames: [0, 3, 2])

7. \*\*4:\*\* Page fault (Replace 2 → Frames: [0, 3, 4])

8. \*\*6:\*\* Page fault (Replace 0 → Frames: [6, 3, 4])

9. \*\*0:\*\* Page fault (Replace 3 → Frames: [6, 0, 4])

10. \*\*2:\*\* Page fault (Replace 4 → Frames: [6, 0, 2])

11. \*\*1:\*\* Page fault (Replace 6 → Frames: [1, 0, 2])

12. \*\*2:\*\* Page hit (Frames: [1, 0, 2])

13. \*\*1:\*\* Page hit (Frames: [1, 0, 2])

14. \*\*2:\*\* Page hit (Frames: [1, 0, 2])

15. \*\*0:\*\* Page hit (Frames: [1, 0, 2])

16. \*\*3:\*\* Page fault (Replace 2 → Frames: [1, 0, 3])

17. \*\*2:\*\* Page fault (Replace 0 → Frames: [1, 2, 3])

18. \*\*1:\*\* Page hit (Frames: [1, 2, 3])

19. \*\*2:\*\* Page hit (Frames: [1, 2, 3])

20. \*\*0:\*\* Page fault (Replace 3 → Frames: [1, 2, 0])

\*\*Total Page Faults:\*\* 14

\*\*Total Page Hits:\*\* 6

### Summary:

| Algorithm | Page Faults | Page Hits |

|-----------|-------------|-----------|

| FIFO | 15 | 5 |

| LRU | 14 | 6 |

Question 4

Let's analyze the completion order of processes P1, P2, and P3 under two scheduling algorithms: First-Come, First-Served (FCFS) and Round Robin with a quantum of 2 time units (RR2).

### Given:

- \*\*Processes:\*\* P1, P2, P3

- \*\*Arrival times:\*\* P1 = 0, P2 = 1, P3 = 3

- \*\*Time units required:\*\* P1 = 5, P2 = 7, P3 = 4

### 1. FCFS (First-Come, First-Served)

Processes are executed in the order of their arrival times.

- \*\*P1\*\* arrives at time 0 and runs for 5 time units (0-5).

- \*\*P2\*\* arrives at time 1 but has to wait for P1 to complete. It starts at time 5 and runs for 7 time units (5-12).

- \*\*P3\*\* arrives at time 3 but has to wait for P1 and P2 to complete. It starts at time 12 and runs for 4 time units (12-16).

\*\*Completion Order:\*\* P1 → P2 → P3

### 2. RR2 (Round Robin with Quantum of 2)

Processes are executed in a cyclic manner, each getting a time slice (quantum) of 2 time units.

| Time | Process Execution | Remaining Time |

|------|-------------------|----------------|

| 0-2 | P1 (executes for 2 units) | P1 = 3, P2 = 7, P3 = 4 |

| 2-4 | P2 (executes for 2 units) | P1 = 3, P2 = 5, P3 = 4 |

| 4-6 | P3 (executes for 2 units) | P1 = 3, P2 = 5, P3 = 2 |

| 6-8 | P1 (executes for 2 units) | P1 = 1, P2 = 5, P3 = 2 |

| 8-10 | P2 (executes for 2 units) | P1 = 1, P2 = 3, P3 = 2 |

| 10-12| P3 (executes for 2 units) | P1 = 1, P2 = 3, P3 = 0 (P3 completes) |

| 12-13| P1 (executes for 1 unit) | P1 = 0, P2 = 3, P3 = 0 (P1 completes) |

| 13-15| P2 (executes for 2 units) | P1 = 0, P2 = 1, P3 = 0 |

| 15-16| P2 (executes for 1 unit) | P1 = 0, P2 = 0, P3 = 0 (P2 completes) |

\*\*Completion Order:\*\* P3 → P1 → P2

### Summary:

| Algorithm | Completion Order |

|-----------|--------------------|

| FCFS | P1 → P2 → P3 |

| RR2 | P3 → P1 → P2 |

Question 5

To calculate the effective memory access time (EMAT) with a TLB (Translation Lookaside Buffer), we can use the following formula based on the TLB hit ratio:

\[

\text{EMAT} = (\text{TLB hit ratio} \times (\text{TLB access time} + \text{Memory access time})) + (\text{TLB miss ratio} \times (\text{TLB access time} + 2 \times \text{Memory access time}))

\]

### Given:

- TLB hit ratio = 80% = 0.8

- TLB miss ratio = 1 - TLB hit ratio = 0.2

- TLB access time = 20 ns

- Memory access time = 100 ns

### Calculation:

1. \*\*Calculate EMAT for TLB hits:\*\*

- Time for TLB hit = TLB access time + Memory access time

\[

= 20 \text{ ns} + 100 \text{ ns} = 120 \text{ ns}

\]

2. \*\*Calculate EMAT for TLB misses:\*\*

- Time for TLB miss = TLB access time + 2 × Memory access time

\[

= 20 \text{ ns} + 2 \times 100 \text{ ns} = 20 \text{ ns} + 200 \text{ ns} = 220 \text{ ns}

\]

3. \*\*Calculate EMAT:\*\*

\[

\text{EMAT} = (0.8 \times 120 \text{ ns}) + (0.2 \times 220 \text{ ns})

\]

\[

= (96 \text{ ns}) + (44 \text{ ns}) = 140 \text{ ns}

\]

### Convert to milliseconds:

\[

\text{EMAT} = 140 \text{ ns} = 140 \times 10^{-9} \text{ s} = 0.000140 \text{ s}

\]

To convert seconds to milliseconds:

\[

\text{EMAT} = 0.000140 \text{ s} \times 1000 = 0.140 \text{ ms}

\]

### Final Answer:

The effective memory access time is \*\*0.140 ms\*\*.

Question 6

To determine the number of context switches needed under the Shortest Remaining Time First (SRTF) scheduling algorithm, let's analyze the arrival and execution of the three CPU-intensive processes.

### Given Processes:

- \*\*P1:\*\* Arrival time = 0, Burst time = 10

- \*\*P2:\*\* Arrival time = 2, Burst time = 20

- \*\*P3:\*\* Arrival time = 6, Burst time = 30

### Step-by-Step Execution:

1. \*\*At time 0:\*\* Only P1 is available. It starts executing.

2. \*\*At time 2:\*\* P1 has executed for 2 time units (remaining time = 8). P2 arrives. Since P1 has the shorter remaining time, it continues.

3. \*\*At time 6:\*\* P1 has executed for 6 time units (remaining time = 4). P3 arrives. P1 still has the shortest remaining time, so it continues.

4. \*\*At time 10:\*\* P1 completes (10 time units used). Now, we compare P2 (remaining time = 20) and P3 (remaining time = 30). P2 has the shortest remaining time and starts executing.

5. \*\*At time 30:\*\* P2 has executed for 20 time units and completes. Now, P3 (remaining time = 30) starts executing.

6. \*\*At time 60:\*\* P3 completes its execution (30 time units used).

### Gantt Chart:

```

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

0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60

```

### Context Switches Calculation:

- \*\*Context Switches:\*\*

- From P1 to P2 (time 10)

- From P2 to P3 (time 30)

\*\*Total Context Switches:\*\* 2

### Summary:

- \*\*Total Context Switches (excluding initial and final):\*\* \*\*2\*\*

- \*\*Gantt Chart:\*\*

```

| P1 | P2 | P3 |

0 10 30 60

```

Question 7

Let me solve this step by step.

1) First, let's break down the execution time for each process:

Process P1 (10 units total):

- First I/O: 20% of 10 = 2 units

- Compute: 70% of 10 = 7 units

- Second I/O: 10% of 10 = 1 unit

Process P2 (20 units total):

- First I/O: 20% of 20 = 4 units

- Compute: 70% of 20 = 14 units

- Second I/O: 10% of 20 = 2 units

Process P3 (30 units total):

- First I/O: 20% of 30 = 6 units

- Compute: 70% of 30 = 21 units

- Second I/O: 10% of 30 = 3 units

2) All processes start with I/O at time 0:

- P1 does I/O for 2 units

- P2 does I/O for 4 units

- P3 does I/O for 6 units

These can overlap.

3) After I/O:

- P1 is ready for compute at t=2

- P2 is ready for compute at t=4

- P3 is ready for compute at t=6

4) Following shortest remaining compute time:

- At t=2, P1 starts computing (has 7 units)

- At t=9, P1 finishes compute and starts final I/O (1 unit)

- At t=9, P2 starts compute (has 14 units)

- At t=23, P2 finishes compute and starts final I/O (2 units)

- At t=23, P3 starts compute (has 21 units)

- At t=44, P3 finishes compute and starts final I/O (3 units)

5) Total time span:

The last process (P3) finishes at t=47

During this time:

- CPU is active during compute periods: 7 + 14 + 21 = 42 units

- CPU is idle during:

\* t=0 to t=2 (2 units)

\* t=9 to t=9 (0 units)

\* t=23 to t=23 (0 units)

\* t=44 to t=47 (3 units)

Total idle time = 5 units

6) Percentage calculation:

(Idle time / Total time) × 100

= (5/47) × 100

≈ 10.64%

Therefore, the CPU remains idle for approximately 10.64% of the time.

Question 8

We are given three processes with arrival times and burst times, and we need to calculate the average waiting time using the \*\*Preemptive Shortest Job First (SJF)\*\* scheduling algorithm.

### Given Data:

| Process | Arrival Time | Burst Time |

|---------|--------------|------------|

| P0 | 0 ms | 9 ms |

| P1 | 1 ms | 4 ms |

| P2 | 2 ms | 9 ms |

### Step-by-Step Execution:

\*\*Preemptive SJF\*\* selects the process with the shortest remaining time, and scheduling occurs at the arrival or completion of a process.

1. \*\*At time 0:\*\*

Only \*\*P0\*\* is available (remaining time = 9 ms), so P0 starts execution.

2. \*\*At time 1:\*\*

\*\*P1\*\* arrives (remaining time = 4 ms). Since P1 has a shorter remaining time than P0 (9 ms), P0 is preempted, and P1 starts execution.

3. \*\*At time 5:\*\*

\*\*P1\*\* completes (4 ms execution). Now, P0 (remaining time = 8 ms) and \*\*P2\*\* (remaining time = 9 ms) are available. Since P0 has the shortest remaining time, P0 resumes execution.

4. \*\*At time 13:\*\*

\*\*P0\*\* completes (8 ms remaining executed). Now, \*\*P2\*\* starts execution since it's the only process left.

5. \*\*At time 22:\*\*

\*\*P2\*\* completes its execution (9 ms executed).

### Gantt Chart:

```

| P0 | P1 | P0 | P2 |

0 1 5 13 22

```

### Completion Times:

- \*\*P0\*\* finishes at time 13.

- \*\*P1\*\* finishes at time 5.

- \*\*P2\*\* finishes at time 22.

### Turnaround Time (TAT):

Turnaround time is the total time taken from the arrival of the process to its completion:

- \*\*TAT for P0:\*\* \( 13 - 0 = 13 \, \text{ms} \)

- \*\*TAT for P1:\*\* \( 5 - 1 = 4 \, \text{ms} \)

- \*\*TAT for P2:\*\* \( 22 - 2 = 20 \, \text{ms} \)

### Waiting Time (WT):

Waiting time is the turnaround time minus the burst time:

- \*\*WT for P0:\*\* \( 13 - 9 = 4 \, \text{ms} \)

- \*\*WT for P1:\*\* \( 4 - 4 = 0 \, \text{ms} \)

- \*\*WT for P2:\*\* \( 20 - 9 = 11 \, \text{ms} \)

### Average Waiting Time:

\[

\text{Average WT} = \frac{WT\_{\text{P0}} + WT\_{\text{P1}} + WT\_{\text{P2}}}{3}

\]

\[

\text{Average WT} = \frac{4 + 0 + 11}{3} = \frac{15}{3} = 5 \, \text{ms}

\]

### Final Answer:

The average waiting time is \*\*5 ms\*\*.

Question 9

To estimate the fraction of time the CPU is busy, we can use the \*\*utilization formula\*\* from queuing theory for a system with a single server (processor):

\[

\text{CPU Utilization} = \frac{\lambda \times \text{Service Time}}{1}

\]

where:

- \(\lambda\) is the arrival rate of processes (number of processes arriving per second),

- \*\*Service Time\*\* is the time required by the CPU to service each process,

- The system has a single CPU, so there's no need to divide by the number of servers.

### Given Data:

- \*\*Service time per process:\*\* 3 seconds

- \*\*Arrival rate (\(\lambda\)):\*\* 10 processes per minute

First, convert the arrival rate into processes per second:

\[

\lambda = \frac{10 \, \text{processes}}{60 \, \text{seconds}} = \frac{1}{6} \, \text{processes per second}

\]

### Now calculate the CPU utilization:

\[

\text{CPU Utilization} = \lambda \times \text{Service Time} = \frac{1}{6} \times 3 = \frac{3}{6} = 0.5

\]

### Final Answer:

The fraction of time the CPU is busy is \*\*0.5\*\*, or \*\*50%\*\*.  
  
Question 10:

Let's break down the problem and calculate the number of bits in both the \*\*logical address\*\* and the \*\*physical address\*\*.

### Given:

- \*\*Logical Address Space:\*\*

- 8 pages

- Each page contains 1024 words

- \*\*Physical Memory:\*\*

- 32 frames

- Each frame can hold a page (of 1024 words)

---

### Part (a): Number of Bits in the Logical Address

To determine the number of bits in the logical address, we need to calculate the total size of the logical address space and figure out how many bits are required to represent it.

1. \*\*Number of Pages:\*\*

There are \*\*8 pages\*\* in the logical address space. To represent these 8 pages, we need:

\[

\text{Number of bits for the page number} = \log\_2(8) = 3 \, \text{bits}

\]

2. \*\*Page Size:\*\*

Each page has \*\*1024 words\*\*. To represent 1024 words (which is the size of each page), we need:

\[

\text{Number of bits for the offset within a page} = \log\_2(1024) = 10 \, \text{bits}

\]

Thus, the total number of bits in the \*\*logical address\*\* is the sum of the bits for the page number and the offset:

\[

\text{Total bits in the logical address} = 3 \, (\text{page bits}) + 10 \, (\text{offset bits}) = 13 \, \text{bits}

\]

---

### Part (b): Number of Bits in the Physical Address

To determine the number of bits in the physical address, we need to calculate how many frames are available in the physical memory and how many bits are needed to represent a frame and its offset.

1. \*\*Number of Frames:\*\*

There are \*\*32 frames\*\* in the physical memory. To represent 32 frames, we need:

\[

\text{Number of bits for the frame number} = \log\_2(32) = 5 \, \text{bits}

\]

2. \*\*Frame Size:\*\*

Each frame holds \*\*1024 words\*\*, just like each page. To represent the offset within a frame, we still need:

\[

\text{Number of bits for the offset within a frame} = 10 \, \text{bits}

\]

Thus, the total number of bits in the \*\*physical address\*\* is the sum of the bits for the frame number and the offset:

\[

\text{Total bits in the physical address} = 5 \, (\text{frame bits}) + 10 \, (\text{offset bits}) = 15 \, \text{bits}

\]

---

### Final Answer:

- \*\*(a) Logical Address:\*\* 13 bits

- \*\*(b) Physical Address:\*\* 15 bits