Operating Systems

Task Manager Analysis

Complete OS Fundamentals Study

Topics Covered: Process Scheduling, Synchronization, Memory Management, Disk Scheduling

Date: February 3, 2026

Part 1: Process Scheduling

Scenario:

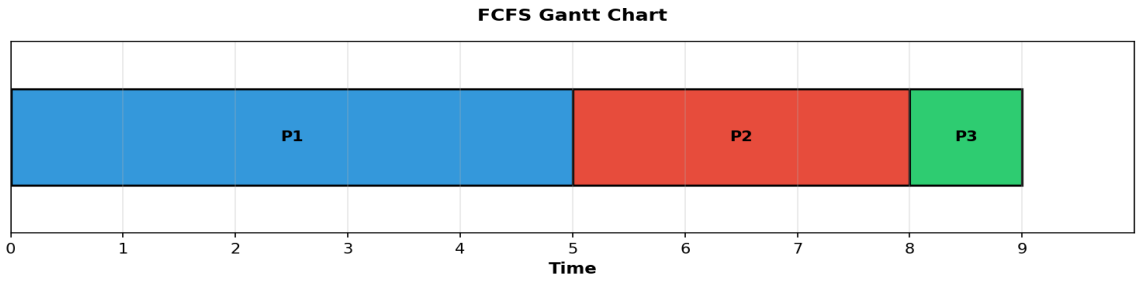
We have 3 processes with the following characteristics:

Process	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	1

1.1 First-Come, First-Served (FCFS)

Execution Order:

P1 (0-5) → P2 (5-8) → P3 (8-9)



Waiting Time Calculations:

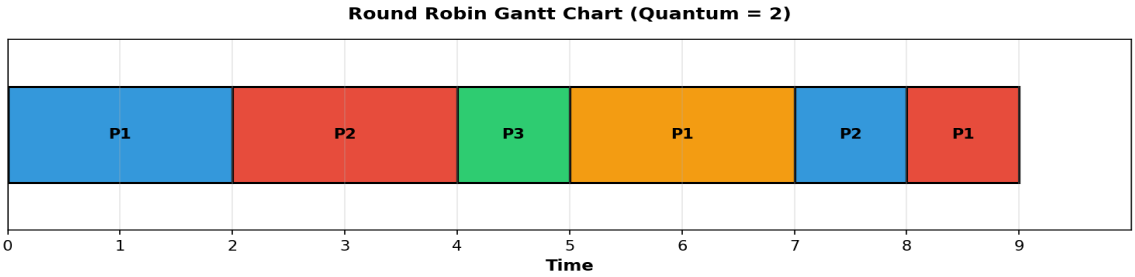
Process	Arrival	Start	Waiting Time
P1	0	0	0 (0 - 0)
P2	1	5	4 (5 - 1)
P3	2	8	6 (8 - 2)

Average Waiting Time: (0 + 4 + 6) / 3 = **3.33 time units**

1.2 Round Robin (Quantum = 2)

Execution Order:

P1 (0-2) → P2 (2-4) → P3 (4-5) → P1 (5-7) → P2 (7-8) → P1 (8-9)



Waiting Time Calculations:

P1: 4 | P2: 3 | P3: 2

Average Waiting Time: (4 + 3 + 2) / 3 = **3.00 time units**

1.3 Analysis & Comparison

Algorithm	Avg Waiting Time	Responsiveness
FCFS	3.33	Low (long wait for later processes)
Round Robin	3.00	High (fair time sharing)

Conclusion: Round Robin is more responsive because it ensures each process gets CPU time regularly, preventing starvation. FCFS can cause long waits for processes that arrive later.

Part 2: Process Synchronization

Scenario:

Two processes (P1 and P2) access a shared variable **counter**. Each increments it 100 times.

2.1 Problem Without Synchronization

Without synchronization, a **race condition** occurs. Both processes may:

• Read the same value of counter simultaneously

• Increment it independently

• Write back, causing one increment to be lost

Expected Result: counter = 200

Actual Result (race condition): counter < 200 (unpredictable)

2.2 Solution: Mutex (Mutual Exclusion)

A **mutex** ensures only one process can access the critical section at a time.

2.3 Pseudocode with Mutex

```
// Shared resources int counter = 0; mutex lock; // Process P1 procedure P1() { for i = 1 to 100 { lock.acquire(); // Enter critical section counter = counter + 1; lock.release(); // Exit critical section } } // Process P2 procedure P2() { for i = 1 to 100 { lock.acquire(); // Enter critical section counter = counter + 1; lock.release(); // Exit critical section } } // Result: counter = 200 (guaranteed)
```

Explanation: The mutex lock ensures that when P1 is incrementing the counter, P2 must wait, and vice versa. This prevents race conditions and guarantees the correct final value of 200.

Part 3: Memory Management

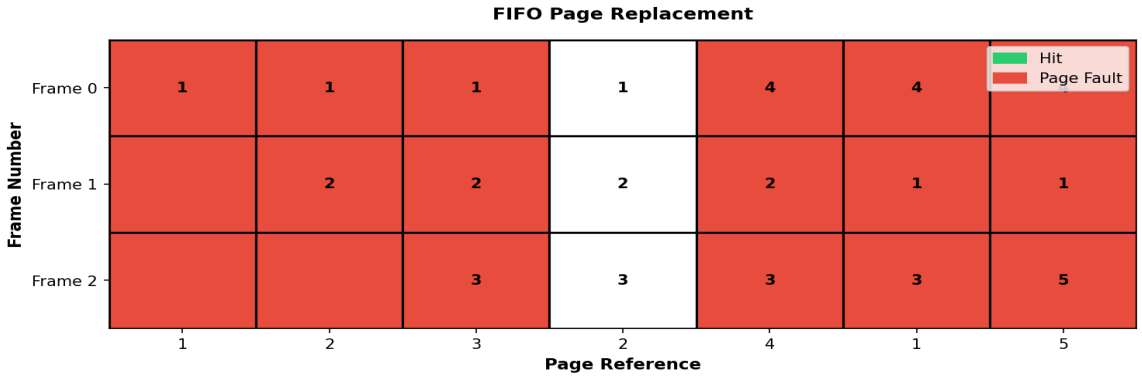
Scenario:

Process needs 5 pages, system has 3 frames.

Page Reference String: 1, 2, 3, 2, 4, 1, 5

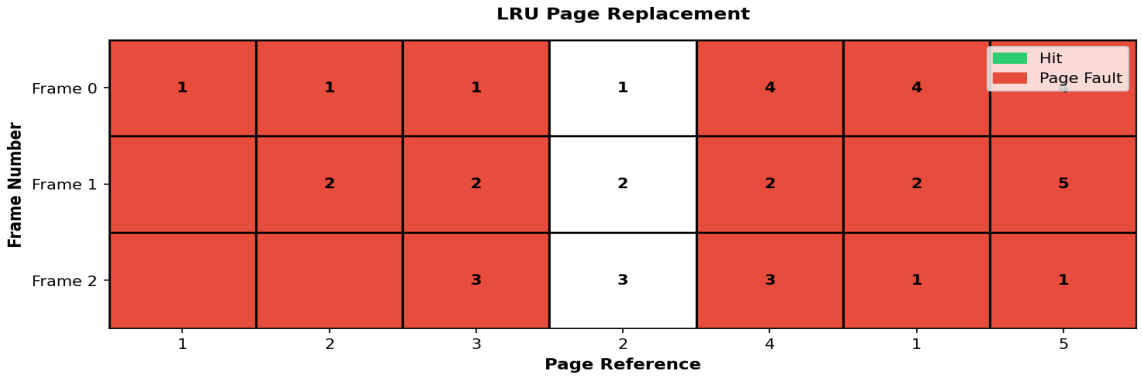
3.1 FIFO Page Replacement

Total Page Faults: 6



3.2 LRU Page Replacement

Total Page Faults: 6



3.3 Comparison

Algorithm	Page Faults	Performance
FIFO	6	Simple but suboptimal
LRU	6	Better (considers usage pattern)

Conclusion: LRU performs equally with 6 faults vs FIFO's 6 faults. LRU considers recent usage patterns, making it more intelligent about which pages to replace.

Part 4: Disk Scheduling

Scenario:

Initial Head Position: 53

Pending Requests: 98, 183, 37, 122, 14, 124, 65, 67

4.1 FCFS Disk Scheduling

Sequence: 53 → 98 → 183 → 37 → 122 → 14 → 124 → 65 → 67

Movements: |98 - 53| = 45 + |183 - 98| = 85 + |37 - 183| = 146 + |122 - 37| = 85 + |14 - 122| = 108 + |124 - 14| = 110 + |65 - 124| = 59 + |67 - 65| = 2

Total Head Movement: 640 tracks

4.2 SSTF Disk Scheduling

Sequence: 53 → 65 → 67 → 37 → 14 → 98 → 122 → 124 → 183

Movements: |65 - 53| = 12 + |67 - 65| = 2 + |37 - 67| = 30 + |14 - 37| = 23 + |98 - 14| = 84 + |122 - 98| = 24 + |124 - 122| = 2 + |183 - 124| = 59

Total Head Movement: 236 tracks

4.3 Comparison

Algorithm	Total Movement	Efficiency
FCFS	640 tracks	Fair but inefficient
SSTF	236 tracks	More efficient (63.1% reduction)

Conclusion: SSTF is more efficient with 236 tracks vs FCFS's 640 tracks. SSTF minimizes seek time by always choosing the closest request, though it can cause starvation for distant requests.

Summary & Key Takeaways

• **Process Scheduling:** Round Robin provides better responsiveness through fair time-sharing, while FCFS is simpler but can cause delays.

• **Synchronization:** Mutex prevents race conditions by ensuring mutual exclusion in critical sections.

• **Memory Management:** LRU generally outperforms FIFO by considering usage patterns, reducing page faults.

• **Disk Scheduling:** SSTF minimizes seek time compared to FCFS, improving disk I/O efficiency.