# Operating Systems Process Management - Unit 4 Assignment

## Q.1 Describe what context switching is in process management.

**Context switching** is the fundamental mechanism in operating systems that allows multiple processes to share a single CPU by alternating between them [1] [2]. It involves storing the current state of a running process so it can be resumed later, and loading the state of another process to begin or continue its execution.

### Key Components of Context Switching:

- **Process Control Block (PCB)**: Stores process state information including program counter, CPU registers, memory allocation, and stack pointer [2]
- **Context**: Comprises process stack, memory address space, virtual memory space, register values, and stack pointer [2]

### Context Switching Steps:

1. Save the current process state to its PCB
2. Update the PCB and move process to appropriate queue (ready, I/O, waiting)
3. Select a new process for execution
4. Update the selected process's PCB to running state
5. Load the new process's context from its PCB
6. Resume execution of the new process [2]

## Q.2 What causes a process to transition from the ready state to the running state?

A process transitions from **ready state to running state** when it is selected by the **CPU scheduler** for execution [3] [4]. This transition occurs when:

### Primary Trigger:

- **CPU becomes available**: The operating system scheduler selects a process from the ready queue based on the scheduling algorithm in use [3]

### Selection Criteria:

- **Scheduling algorithm**: Determines which process gets selected (FCFS, SJF, Priority, Round Robin, etc.) [5]
- **Process priority**: Higher priority processes may be selected first
- **Arrival time**: In FCFS, processes are selected based on arrival order
- **Burst time**: In SJF, shortest processes are selected first

The scheduler ensures optimal CPU utilization by selecting the most appropriate process from the ready queue according to the system's scheduling policy[6].

## Q.3 What is process scheduling and why is it important?

**Process scheduling** is the method by which the operating system decides which process should be executed by the CPU at any given time[5] [7]. It manages the allocation of CPU time among multiple competing processes.

## Why Process Scheduling is Important:

**1. Resource Optimization**

- Maximizes CPU utilization and system throughput[5] [8]
- Prevents CPU from remaining idle when processes are available

**2. System Performance**

- Minimizes waiting time and turnaround time for processes[5]
- Improves overall system responsiveness and efficiency

**3. Fairness**

- Ensures all processes get fair access to CPU resources[6]
- Prevents process starvation where some processes never get executed

**4. Multitasking Support**

- Enables multiple processes to run apparently simultaneously[9]
- Essential for modern multiprogramming environments

**5. Priority Management**

- Allows important processes to be executed with higher priority[5]
- Supports real-time systems with critical timing requirements

## Q.4 Given a scenario, describe which type of scheduling algorithm should be used and justify your choice.

The choice of scheduling algorithm depends on the specific system requirements and environment:

## Batch Processing Systems

**Algorithm**: First-Come, First-Served (FCFS)
**Justification**: Simple implementation, suitable for systems with similar execution times and no user interaction requirements[5] [6]

## Interactive/Time-Sharing Systems

**Algorithm**: Round Robin (RR)
**Justification**: Ensures fair CPU time distribution, good response time for interactive applications, prevents starvation[10] [11]

## Real-Time Systems

**Algorithm**: Priority Scheduling (Preemptive)
**Justification**: Critical tasks can preempt lower priority ones, meets timing constraints essential for real-time operations[5] [12]

## Mixed Workload Environments

**Algorithm**: Multilevel Queue Scheduling
**Justification**: Different types of processes can use appropriate algorithms, balances efficiency with fairness

## Short Job Environments

**Algorithm**: Shortest Job First (SJF)
**Justification**: Minimizes average waiting time, optimal for environments with predictable, short execution times[5] [6]

## Q.5 Explain the concept of multithreading and its advantages.

**Multithreading** is the ability of an operating system to execute multiple threads of a process concurrently, allowing different parts of a program to run simultaneously while sharing the same resources[13] [14].

## Key Concepts:

- **Thread**: Lightweight sub-process or execution path within a program[13]
- **Concurrency**: Multiple threads can execute simultaneously on multi-processor systems[14]
- **Resource Sharing**: Threads within a process share memory, files, and other resources[13]

## Advantages of Multithreading:

#### 1. Improved Responsiveness

- Applications remain responsive even when one thread is blocked[13] [14]
- Users can interact with one part while another part processes data

#### 2. Resource Sharing

- Threads automatically share memory and resources of their parent process[13]
- More efficient than inter-process communication

#### 3. Economic Benefits

- Creating threads requires less overhead than creating processes[13] [14]
- Thread management is more cost-effective than process management

#### 4. Enhanced Performance

- Better utilization of multiprocessor architectures[13] [15]
- Parallel execution on multiple CPUs increases throughput

**5. Improved Communication**

- Thread synchronization provides better inter-process communication[13] [14]
- High-bandwidth, low-latency communication within applications

**6. Scalability**

- Applications can scale better with available system resources[14]
- Effective utilization of system capabilities

## Q.6 Differentiate between preemptive and non-preemptive scheduling.

| Aspect | Preemptive Scheduling | Non-Preemptive Scheduling |
|---|---|---|
| **Process Control** | OS can interrupt running processes [16] [17] | Process runs until completion or I/O [16] [18] |
| **Resource Allocation** | Resources allocated for limited time [16] [19] | Process holds resources until termination [18] [19] |
| **Interruption** | Processes can be interrupted mid-execution [16] | No interruption until process completes [16] |
| **Context Switching** | Frequent context switches, higher overhead [16] [17] | Less frequent switching, lower overhead [16] |
| **Response Time** | Lower response time, better for interactive systems [16] [12] | Higher response time, suitable for batch processing [18] |
| **CPU Utilization** | Higher CPU utilization [18] [17] | Lower CPU utilization [18] |
| **Fairness** | More fair, prevents monopolization [12] | Less fair, long processes can delay others [12] |
| **Implementation** | More complex to implement [17] [12] | Simpler implementation [17] |
| **Cost** | Higher cost due to overhead [18] | Lower cost, no scheduling overhead [18] |
| **Examples** | Round Robin, Priority Scheduling, SRTF [16] [12] | FCFS, SJF (non-preemptive) [16] |
| **Best For** | Multitasking, real-time systems [12] | Batch processing, simple systems [12] |

## Q.7 Describe the First Come First Served (FCFS) scheduling algorithm.

**First Come First Served (FCFS)** is the simplest CPU scheduling algorithm that executes processes in the order they arrive in the ready queue[5] [6].

### Characteristics:

- **Non-preemptive**: Once a process starts execution, it runs to completion[6]
- **FIFO principle**: First process to arrive gets served first[6]
- **Simple implementation**: Easy to understand and implement[5]

### How FCFS Works:

1. Processes are queued in order of arrival
2. CPU executes the first process in the queue
3. Process runs until completion or I/O operation
4. Next process in queue gets CPU time

### Advantages:

- **Simplicity**: Easy to implement and understand[5] [6]
- **Fair**: Processes are served in arrival order[6]
- **No starvation**: Every process eventually gets executed
- **Low overhead**: Minimal scheduling overhead

### Disadvantages:

- **Convoy effect**: Short processes wait for long processes[5] [6]
- **Poor turnaround time**: Average waiting time can be high[5]
- **Inefficient**: Not suitable for interactive systems
- **No priority consideration**: Important processes may wait longer

### Best Use Cases:

- Batch processing systems with similar execution times[5]
- Simple systems where arrival order is the primary concern
- Non-interactive environments

## Q.8 Explain the Shortest Job First (SJF) scheduling algorithm.

**Shortest Job First (SJF)** is a scheduling algorithm that selects the process with the shortest burst time (execution time) for execution next[5] [6].

### Types of SJF:

1. **Non-preemptive SJF**: Process runs to completion once started
2. **Preemptive SJF (SRTF)**: Shortest Remaining Time First - can preempt currently running process[5]

### How SJF Works:

1. Scheduler examines all processes in ready queue
2. Selects process with minimum burst time
3. Executes selected process (completely in non-preemptive, or until shorter job arrives in preemptive)
4. Repeats selection process

### Advantages:

- **Optimal average waiting time**: Minimizes average waiting time for given set of processes[5] [6]
- **High throughput**: More short processes complete quickly
- **Efficient resource utilization**: Better CPU utilization than FCFS

### Disadvantages:

- **Starvation problem**: Long processes may never execute if short processes keep arriving[5] [6]
- **Prediction difficulty**: Hard to predict actual burst time in practice
- **Not suitable for interactive systems**: May cause poor response time for long processes

### Best Use Cases:

- Batch processing with known execution times[5]
- Environments where burst time can be accurately predicted
- Systems prioritizing quick turnaround for short tasks

## Q.9 How does the Round Robin (RR) scheduling algorithm work? Give an example.

**Round Robin (RR)** is a preemptive scheduling algorithm that assigns each process a fixed time slice (quantum) and cycles through processes in a circular queue[10] [11].

### How Round Robin Works:

1. Each process gets a fixed time quantum (e.g., 2 seconds)
2. Process executes for its time slice
3. If process completes within time slice, it terminates
4. If time expires, process is preempted and moved to end of ready queue
5. Next process in queue gets CPU time

### Example:

**Given processes with time quantum = 2 seconds:**

| Process | Burst Time |
|---|---|
| P1 | 4 |
| P2 | 3 |
| P3 | 5 |

**Execution Timeline:**

- **Time 0-2**: P1 executes (remaining: 2), moves to end of queue
- **Time 2-4**: P2 executes (remaining: 1), moves to end of queue
- **Time 4-6**: P3 executes (remaining: 3), moves to end of queue
- **Time 6-8**: P1 executes (remaining: 0), completes
- **Time 8-9**: P2 executes (remaining: 0), completes
- **Time 9-11**: P3 executes (remaining: 1), moves to end
- **Time 11-12**: P3 executes (remaining: 0), completes[10]

### Advantages:

- **Fair allocation**: Every process gets equal CPU time[10] [11]
- **No starvation**: All processes eventually execute
- **Good response time**: Suitable for interactive systems[11]
- **Predictable**: Easy to estimate completion times

### Disadvantages:

- **Context switching overhead**: Frequent switching reduces efficiency[10] [20]
- **Performance depends on quantum size**: Too small increases overhead, too large approaches FCFS[20]
- **Higher waiting time**: May have longer average waiting time than SJF[10]

## Q.10 Discuss how priority scheduling can cause starvation and how it can be mitigated.

**Priority Scheduling** assigns priority levels to processes, with higher priority processes executed before lower priority ones[5]. However, this can lead to **starvation**.

### What is Starvation?

**Starvation** occurs when low-priority processes are indefinitely delayed because higher-priority processes continuously arrive and get executed first[5]. The low-priority process may never get CPU time.

### How Priority Scheduling Causes Starvation:

1. **Continuous arrival of high-priority processes**: New high-priority processes keep entering the system
2. **Indefinite postponement**: Low-priority processes remain in ready queue indefinitely
3. **Resource monopolization**: High-priority processes dominate CPU usage
4. **System unfairness**: Some processes never get chance to execute

### Mitigation Techniques:

**1. Aging Technique**

- Gradually increase priority of waiting processes over time[5]
- Older processes eventually gain high enough priority to execute
- Ensures eventual execution of all processes

**2. Priority Inheritance**

- Temporarily boost priority of low-priority processes holding resources needed by high-priority processes
- Prevents priority inversion problems

**3. Multilevel Feedback Queues**

- Use multiple priority levels with different scheduling algorithms
- Allow processes to move between priority levels based on behavior

**4. Time Slicing with Priority**

- Combine priority scheduling with round-robin within same priority level
- Ensures processes at same priority get fair CPU time

**5. Priority Bounds**

- Set maximum waiting time limits
- Automatically boost priority after certain waiting period

**6. Mixed Scheduling Algorithms**

- Use different algorithms for different priority classes

- Balance priority requirements with fairness considerations

## Q.11 Outline the key features of process scheduling in UNIX operating systems.

UNIX operating systems implement sophisticated **multilevel feedback queue scheduling** with several distinctive features[21] [22]:

## Key Features:

**1. Multilevel Priority Queues**

- Multiple priority levels (typically 0-127 or 0-255)
- Higher numbers indicate lower priority
- Separate queues for different priority classes

**2. Dynamic Priority Adjustment**

- Process priorities change based on behavior and resource usage[22]
- CPU-intensive processes get lower priority over time
- I/O-bound processes maintain higher priority

**3. Time Slicing**

- Each process gets time quantum based on priority level
- Higher priority processes get longer time slices
- Preemptive scheduling with time slice expiration

**4. Process States**

- **Runnable**: Ready to execute, in priority queue
- **Sleeping**: Waiting for event or resource[22]
- **Zombie**: Terminated but not yet cleaned up
- **Stopped**: Suspended by signal

**5. Nice Values**

- User-controllable priority adjustment mechanism
- Range from -20 (highest) to +19 (lowest priority)
- Allows users to influence process scheduling

**6. Real-time Scheduling Classes**

- Support for real-time processes with guaranteed response times
- FIFO and Round-Robin real-time scheduling policies
- Higher priority than normal time-sharing processes

**7. Load Balancing**

- In multiprocessor systems, distribute processes across CPUs
- Migration of processes between processors for optimal performance

**8. Interactive Process Detection**

- System identifies interactive processes automatically
- Provides better response time for user-interactive applications

**9. Aging Mechanism**

- Prevents starvation by gradually increasing priority of waiting processes
- Ensures long-waiting processes eventually get CPU time

**10. Context Switching Optimization**

- Efficient context switching mechanisms
- Minimal overhead during process transitions[21]
  [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40]

⁂


1. https://www.geeksforgeeks.org/operating-systems/states-of-a-process-in-operating-systems/
2. https://byjus.com/gate/process-state-in-operating-system-notes/
3. https://www.alibabacloud.com/tech-news/a/shueduler/gv69skl0pm-the-science-behind-scheduling-algorithms
4. https://curatepartners.com/blogs/skills-tools-platforms/understanding-scheduling-algorithms-optimizing-system-performance-and-efficiency/
5. https://www.geeksforgeeks.org/operating-systems/context-switch-in-operating-system/
6. https://www.techtarget.com/whatis/definition/context-switch
7. https://www.scaler.com/topics/operating-system/context-switching-in-os/
8. https://en.wikipedia.org/wiki/Context_switch
9. https://www.tutorialspoint.com/operating_system/os_context_switching.htm
10. https://www.geeksforgeeks.org/operating-systems/preemptive-and-non-preemptive-scheduling/
11. https://www.geeksforgeeks.org/operating-systems/difference-between-preemptive-and-non-preemptive-cpu-scheduling-algorithms/
12. https://www.multisoftvirtualacademy.com/blog/common-advantages-and-disadvantages-of-multithreading-in-java
13. https://data-flair.training/blogs/round-robin-scheduling-algorithm/
14. https://www.scaler.com/topics/round-robin-scheduling-in-os/
15. https://www.studytonight.com/operating-system/round-robin-scheduling
16. https://docs.oracle.com/cd/E19455-01/806-3461/6jck06gqj/index.html
17. https://www.shiksha.com/online-courses/articles/threads-and-multi-threading-operating-system/
18. https://www.geeksforgeeks.org/operating-systems/benefits-of-multithreading-in-operating-system/
19. https://unstop.com/blog/multithreading-in-os
20. https://byjus.com/gate/difference-between-preemptive-and-non-preemptive-scheduling/
21. https://www.geeksforgeeks.org/operating-systems/process-schedulers-in-operating-system/
22. https://www.geeksforgeeks.org/operating-systems/cpu-scheduling-in-operating-systems/
23. https://byjus.com/gate/context-switching-in-os-notes/

24. https://mohitmishra786.github.io/exploring-os/src/day-02-process-states-and-transitions.html
25. https://docs.oracle.com/cd/E19683-01/806-4125/psched-16/index.html
26. https://testbook.com/question-answer/consider-the-following-statements-about-process-st--5e9d9644f60d5d2560943713
27. https://unstop.com/blog/scheduling-algorithms-in-operating-system
28. https://www.geeksforgeeks.org/operating-systems/introduction-of-process-management/
29. https://www.youtube.com/watch?v=lD3llXoNwlo
30. https://www.tutorialspoint.com/benefits-of-multithreading-in-operating-system
31. https://www.b12.io/resource-center/online-scheduling/should-you-use-round-robin-scheduling.html
32. https://testbook.com/key-differences/difference-between-preemptive-and-non-preemptive-scheduling
33. https://www.geeksforgeeks.org/operating-systems/advantages-and-disadvantages-of-various-cpu-scheduling-algorithms/
34. https://drbtaneja.com/preemptive-vs-non-preemptive-scheduling/
35. https://www.geeksforgeeks.org/operating-systems/round-robin-scheduling-in-operating-system/
36. https://www.tutorialspoint.com/difference-between-preemptive-and-non-preemptive-scheduling-in-os
37. https://www.geeksforgeeks.org/operating-systems/multithreading-in-operating-system/
38. https://www.linkedin.com/advice/0/how-does-round-robin-algorithm-schedule-tasks-irh4c
39. https://www.geeksforgeeks.org/operating-systems/six-state-process-model-in-operating-system/
40. https://www.redwood.com/article/job-scheduling-algorithms/