**The University of Azad Jammu and Kashmir, Muzaffarabad**



|  |  |
| --- | --- |
| **Submitted to:** | **Engr. Sidra Rafique** |
| **Course Title:** | **Operating System** |
| **Course Code:** | **SE-3205** |
| **Session:** | **2022-26** |
| **Semester:** | **6th** |
| **Roll No:** | **2022-SE-01** |
| **Submitted from:** | **Khurram Farman** |
| **Lab Task No** | **8** |
|  |  |

**Bachelors of Science in Software Engineering (2022-26)**

**Department of Software Engineering**

# ****CPU Scheduling Algorithms: A Comparative Study****

## ****Introduction****

CPU scheduling is a crucial aspect of operating system design that determines the order in which processes are executed. Efficient scheduling enhances system performance and ensures better utilization of CPU resources. Several scheduling algorithms exist, each with unique advantages, disadvantages, and performance implications. This report compares four widely used algorithms: First-Come-First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR).

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

### Working Principle

* Processes are executed in the order they arrive in the ready queue.
* It follows the **non-preemptive** approach (once a process starts, it runs until completion).

### Advantages

* Simple to understand and implement.
* Fair to processes since jobs are handled in arrival order.

### Disadvantages

* **Poor average waiting time** if short processes are stuck behind long ones (convoy effect).
* Not suitable for interactive systems.

### Suitable Scenarios

* Best for batch systems where all processes have similar burst times.
* Useful when fairness (arrival order execution) is more important than responsiveness.

### Performance Impact

* **Throughput**: Can be low due to long processes delaying others.
* **Turnaround time**: Highly variable; short jobs may suffer.
* **Waiting time**: High for short processes arriving after long jobs.
* **Response time**: Poor, especially for interactive processes.

## ****2. Shortest Job First (SJF)****

### Working Principle

* Selects the process with the **shortest burst time**.
* Can be **non-preemptive** or **preemptive** (Shortest Remaining Time First).

### Advantages

* Minimizes average waiting time and turnaround time.
* Provides optimal throughput when job lengths are predictable.

### Disadvantages

* Requires prior knowledge or estimation of burst times, which may be inaccurate.
* Risk of **starvation** for longer processes if short jobs keep arriving.

### Suitable Scenarios

* Suitable for batch processing environments where job lengths can be estimated.
* Not ideal for real-time or interactive systems.

### Performance Impact

* **Throughput**: High, since shorter jobs complete quickly.
* **Turnaround time**: Low on average.
* **Waiting time**: Optimized for short processes.
* **Response time**: Good for short jobs, poor for long jobs.

## ****3. Priority Scheduling****

### Working Principle

* Each process is assigned a **priority value**.
* CPU is allocated to the process with the highest priority.
* Can be **preemptive** or **non-preemptive**.

### Advantages

* Useful when some processes are more critical than others.
* Flexible: priorities can be based on memory, I/O needs, or importance.

### Disadvantages

* Risk of **starvation** for lower-priority processes.
* Priority inversion (low-priority process holding resources needed by high-priority process).

### Suitable Scenarios

* Real-time systems where some tasks are more time-sensitive.
* Systems requiring differentiation between user and system processes.

### Performance Impact

* **Throughput**: Depends on priority distribution.
* **Turnaround time**: Good for high-priority processes, bad for low-priority ones.
* **Waiting time**: Can be long for low-priority processes.
* **Response time**: Excellent for high-priority processes.

## ****4. Round Robin (RR)****

### Working Principle

* Each process is assigned a **time quantum** (fixed CPU slice).
* Processes are executed in a cyclic order; if not finished, they return to the queue.

### Advantages

* Fair allocation; no process starves.
* Good response time for interactive systems.

### Disadvantages

* Performance heavily depends on **time quantum** selection:
  + Too small: excessive context switching (overhead).
  + Too large: behaves like FCFS.

### Suitable Scenarios

* Time-sharing systems and interactive environments.
* Systems requiring fairness among all processes.

### Performance Impact

* **Throughput**: Moderate, as frequent context switching may reduce efficiency.
* **Turnaround time**: Can be high for long jobs.
* **Waiting time**: Reasonable, since every process gets CPU time fairly.
* **Response time**: Good for all processes; suitable for interactive tasks.

## ****Comparative Analysis****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ****Algorithm**** | ****Preemption**** | ****Fairness**** | ****Starvation Risk**** | ****Best for**** |
| FCFS | Non-preemptive | Fair (arrival order) | None | Batch systems |
| SJF | Both possible | Biased to short jobs | High (long jobs starve) | Batch jobs with predictable burst times |
| Priority | Both possible | Biased to high-priority | High (low-priority starve) | Real-time & critical systems |
| RR | Preemptive | High | None | Time-sharing, interactive systems |

## ****Conclusion****

Each CPU scheduling algorithm has strengths and weaknesses.

* **FCFS** is simple but inefficient for mixed workloads.
* **SJF** provides optimal waiting time but risks starving long processes.
* **Priority Scheduling** ensures important tasks finish quickly but can neglect lower-priority ones.
* **RR** balances fairness and responsiveness but may suffer from overhead if time quantum is poorly chosen.

The choice of scheduling algorithm depends on the **system’s goals**—whether fairness, responsiveness, or efficiency is prioritized. No single algorithm is universally optimal; instead, system designers select algorithms based on workload characteristics and performance requirements.