Vivekanand Education Society's Institute Of Technology
Department Of Information Technology
DSA mini Project
A.Y. 2025-26

# Title: Job Scheduling System using Priority Queue

Sustainability goal: Optimize computational resources by ensuring jobs are executed in an efficient, priority-based manner, reducing wasted processing time and energy consumption.

Domain: Data structures and algorithms

Member: Unmesh Bhangale
Roll no : 6
DIV: D10B

1 NO POVERTY

2 ZERO HUNGER

3 GOOD HEALTH AND WELL-BEING

4 QUALITY EDUCATION

5 GENDER EQUALITY

6 CLEAN WATER AND SANITATION

7 AFFORDABLE AND CLEAN ENERGY

8 DECENT WORK AND ECONOMIC GROWTH

9 INDUSTRY, INNOVATION AND INFRASTRUCTURE

10 REDUCED INEQUALITIES

11 SUSTAINABLE CITIES AND COMMUNITIES

# THE GLOBAL GOALS

For Sustainable Development

12 RESPONSIBLE CONSUMPTION AND PRODUCTION

13 CLIMATE ACTION

14 LIFE BELOW WATER

15 LIFE ON LAND

16 PEACE AND JUSTICE STRONG INSTITUTIONS

17 PARTNERSHIPS FOR THE GOALS

# Content

1. Introduction to the Project
2. Problem Statement
3. Objectives of the Project
4. Scope of the Project
5. Requirements of the System (Hardware, Software)
6. ER Diagram of the Proposed System
7. Data Structure & Concepts Used
8. Algorithm Explanation
9. Time and Space Complexity
10. Front End
11. Implementation
12. Gantt Chart

13. Test Cases

14.Challenges and Solutions

15. Future Scope

16. Code

17. Output Screenshots

18. Conclusion

19. References (in IEEE Format)

# Introduction to Project

Job Scheduling is a fundamental concept in computer science and operating systems. It deals with **deciding the order in which jobs/processes should be executed** by the system. A job can be a program, a task, or a request waiting for execution.

In a **priority scheduling system**, every job is assigned a **priority value**, and jobs with **higher priority** are executed before jobs with lower priority. This ensures that urgent or critical tasks get processed first.

My project simulates a **Job Scheduling System** using the **Priority Queue data structure**, where we can add jobs, display them, and execute them in priority order.

# Problem Statement

**JOB SHEDULING SYSTEM USING PRIORITY QUEUE**

n traditional scheduling methods like **First Come First Serve (FCFS)**, jobs are executed in the order they arrive. However, this does not consider the importance or urgency of jobs. For example, a critical job may be delayed if it arrives after several lower-priority jobs.

This leads to **inefficiency and delays in execution**. Therefore, a system is needed that can handle jobs based on **priority values** rather than just arrival time.

# Objectives of the project

- To design and develop a **menu-driven scheduling system**.

- To allow insertion of jobs with different **names and priorities**.

- To maintain a **priority queue** for execution.

- To provide functions to **add jobs, display the queue, and execute jobs**.

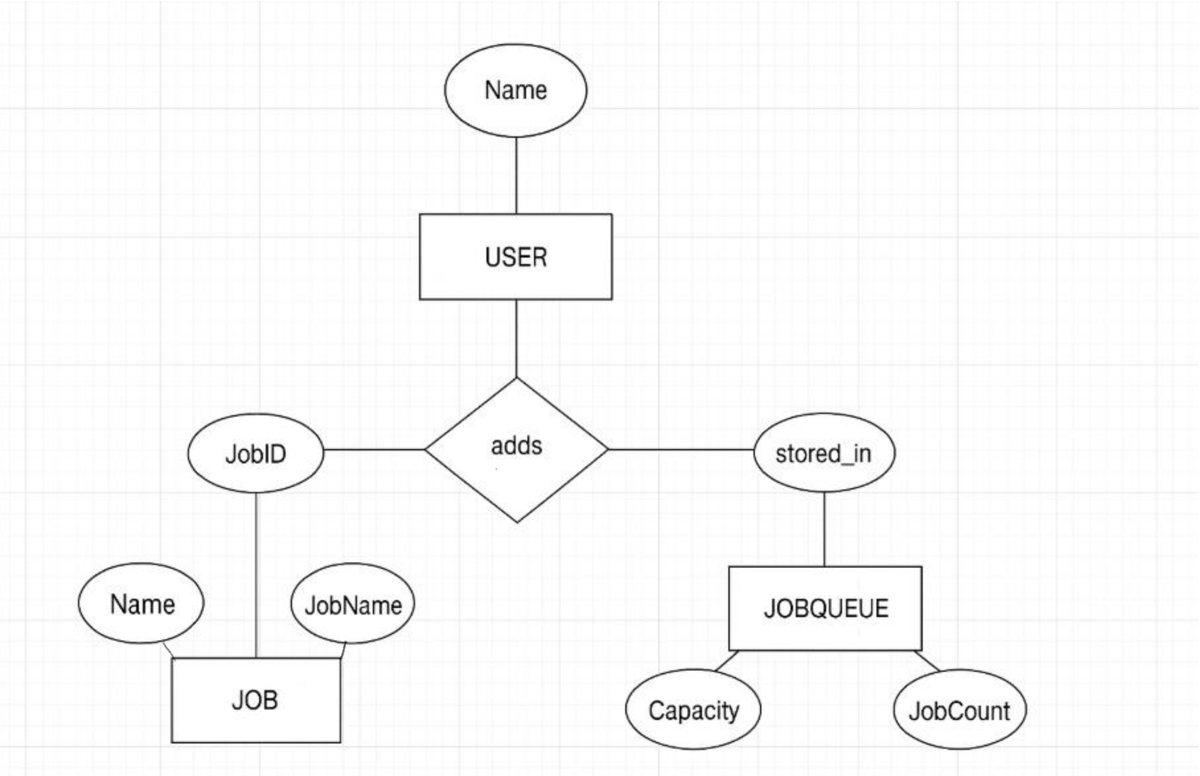
- To demonstrate the application of **data structures in system scheduling**.

# Requirements of the system (Hardware, software)

**Hardware Requirements:**
Processor: Intel i3 or above
RAM: Minimum 4GB
Storage: 200MB free space

**Software Requirements:**
Operating System: Windows / Linux
Compiler: GCC / Turbo C
Language: C
Editor: VS Code / Code::Blocks

# ER diagram of the proposed system



# Front End

- Console-based interface using menu-driven program.
- Input: Job details (Name + Priority).
- Output: Job Queue status and execution messages.

# Implementation

- **Functions:** addJob(), displayJobs(), executeJob().
- **Control:** switch-case driven main menu.
- **Data Structure:** Array-based priority queue.

# Gantt Chart

- Higher priority = earlier execution
- If two jobs have the same priority, they are executed in the order they were inserted.

Let’s assume the user entered the following jobs:

| JobID | JobName | Priority |
|---|---|---|
| 1 | A | 3 |
| 2 | B | 5 |
| 3 | C | 2 |
| 4 | D | 4 |

**Execution Order (based on priority):**

- Job B (Priority 5)
- Job D (Priority 4)
- Job A (Priority 3)
- Job C (Priority 2)

```
Time →   0    1    2    3
        |----|----|----|----|
Job →    B    D    A    C
```

# Test cases

| Test Case | Input | Expected Output |
|---|---|---|
| **1** | Add 3 jobs: J1(priority=5, burst=4, arrival=0), J2(priority=8, burst=3, arrival=1), J3(priority=3, burst=2, arrival=2) | Queue order → J2 → J1 → J3; Execution order same; Waiting times and turnaround times calculated correctly |
| **2** | Add jobs with same priority (J1, J2 both priority=7) | Tie resolved by arrival order (the one added first runs first) |
| **3** | Execute when queue is empty | System should print ⚠ "No jobs to execute" |
| **4** | Add more than MAX (100) jobs | System should print ⚠ "Queue full! Cannot add more jobs." |
| **5** | Search for job ID that does not exist | Should print ❌ "Job not found" |
| **6** | Job arrives later than current time (arrival=5 but CPU idle at 0) | CPU waits until arrival=5 before executing job |

# Challenges and solutions

| Challenge | Explanation | Solution |
|---|---|---|
| **Maintaining priority order efficiently** | Insertion in array requires shifting elements (O(n)) | Use a **Heap-based Priority Queue** for O(log n) insertion |
| **Handling jobs with same priority** | Ambiguity which job to run first | Use **FCFS (First Come First Serve)** rule to break ties |
| **CPU idle time** | When no jobs are available before current time | Add check for **arrival time** and keep CPU idle until a job arrives |
| **Scalability** | Array-based queue limited to MAX=100 | Use **dynamic memory allocation (linked list / heap)** for larger queues |
| **Preemption** | Current system is non-preemptive (once job starts, it finishes) | Extend to **Preemptive Priority Scheduling** for real OS-like behavior |

# Future Scope

**Preemptive Scheduling**
Implement real-time interruption when a higher-priority job arrives mid-execution.

**Multi-Processor Scheduling**
Extend to handle **multiple CPUs** (parallel job execution).

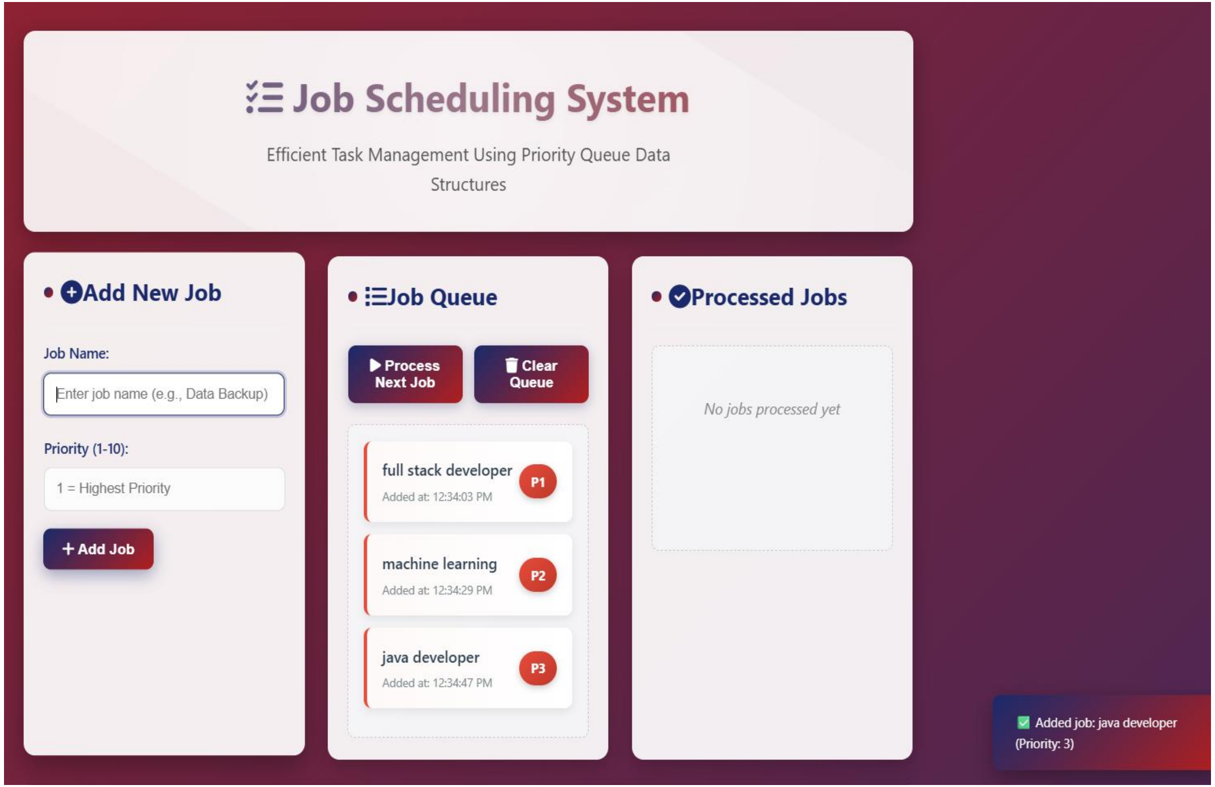
**Different Scheduling Algorithms**
Add support for **Round Robin**, **SJF (Shortest Job First)**, **Multilevel Queue Scheduling**.

**Graphical User Interface (GUI)**
Create a visualization of the Gantt chart (timeline) for better understanding.
**Persistent Storage**

Store jobs in a database so system state is not lost when the program ends.

**Integration with Cloud / Distributed Systems**
Extend scheduler to allocate jobs across **distributed servers** (like Kubernetes job scheduling).

# OUTPUT SCREENSHOT



## Job Scheduling System

Efficient Task Management Using Priority Queue Data Structures

**Add New Job**

Job Name: Enter job name (e.g., Data Backup)

Priority (1-10): 1 = Highest Priority

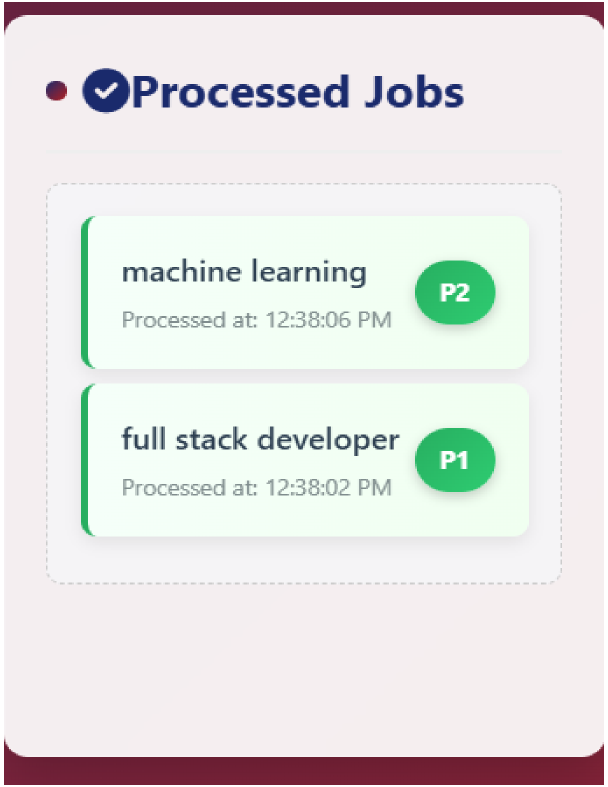
**Job Queue**

- full stack developer — P1 — Added at: 12:34:03 PM
- machine learning — P2 — Added at: 12:34:29 PM
- java developer — P3 — Added at: 12:34:47 PM

**Processed Jobs**

*No jobs processed yet*

✅ Added job: java developer (Priority: 3)



**Processed Jobs**

- machine learning — P2 — Processed at: 12:38:06 PM
- full stack developer — P1 — Processed at: 12:38:02 PM

# Conclusion

The project demonstrates how **priority scheduling** can be effectively implemented using the **priority queue data structure**.

It shows the importance of job prioritization in real-world systems and provides a base that can be extended into more complex scheduling algorithms used in operating systems.

# References

- E. Horowitz, S. Sahni, “Fundamentals of Data Structures in C,” Computer Science Press.
- GeeksforGeeks, “Priority Queue in Data Structures,”. Available: https://www.geeksforgeeks.org
- TutorialsPoint, “Job Scheduling Algorithms,” [Online].
- A. Silberschatz, P.B. Galvin, G. Gagne, “Operating System Concepts,” Wiley, 9th Edition.