



SR University, Ananthsagar, Warangal, Telangana-506371

Multi-Level Queue Scheduling

A Mini Project report submitted
in partial fulfillment of requirement for the award of degree

BACHELOR OF TECHNOLOGY

in

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

by

K.Jayshree	(2403A510B1)
T.Srivalli Reddy	(2403A510B2)
N.Akshaya	(2403A510B8)
G.Meghana	(2403A510C1)

Under the guidance of

Dr.Shanker Chandre

Assistant Professor

School of Computer Science And Artificial Intelligence



CERTIFICATE

This is to certify that this project entitled **"Multi-Level Queue Scheduling"** is the bonafied work carried out by **K.Jayshree, T.Srivalli Reddy, N.Akshaya, G.Meghana** as a Mini Project for the partial fulfillment to award the degree **BACHELOR OF TECHNOLOGY** in **School of Computer Science and Artificial Intelligence** during the academic year 2024-2025 under our guidance and Supervision.

Dr. Shanker Chandre

Assistant Professor,

School of Computer Science And Artificial Intelligence,

SR University, Anathasagar, Warangal



ACKNOWLEDGEMENT

We owe an enormous debt of gratitude to our Major Project guide **Dr. Shanker Chandre, Assistant Professor** as well as Head of the School of CS&AI , **Dr. M.Sheshikala, Professor** and Dean of the School of CS&AI, **Dr.Indrajeet Gupta Professor** for guiding us from the beginning through the end of the Mini Project with their intellectual advices and insightful suggestions. We truly value their consistent feedback on our progress, which was always constructive and encouraging and ultimately drove us to the right direction.

We express our thanks to project co-ordinators **Dr.Shanker Chandre Asst. Prof.**, for their encouragement and support.

Finally, we express our thanks to all the teaching and non-teaching staff of the department for their suggestions and timely support.

Table of Contents	Page Number
1. Objective	5
2. Introduction	6
3. Overview of the Work	
3.1.Problem description	7 - 9
3.2.Software requirements	
3.3.Hardware requirements	
4. System design	
4.1.Description of Modules/Programs	10 - 15
4.2. Methodology	
4.3.System Architecture	
5. Implementation	
5.1.Source code	16 - 22
5.2.Execution screenshots	
6. Conclusion and future Directions	23 - 24
7. References	25

Objective

- The primary objective of this project is to design and implement a Multi-Level Queue Scheduling algorithm that efficiently manages process execution based on priority levels.
- This project aims to:
 - Develop a graphical user interface (GUI)-based process scheduling simulator using Python and Tkinter.
 - Implement a hybrid scheduling approach combining Round Robin (RR) and First-Come-First-Serve (FCFS) techniques to optimize CPU utilization and response time.
 - Provide real-time process input functionality, allowing users to dynamically enter process details such as Arrival Time (AT), Burst Time (BT), and Priority (P).
 - Implement an automated queue classification system that categorizes processes into different priority levels based on median priority values.
 - Visualize the execution order and completion times using a Gantt chart, making the scheduling process intuitive and easy to understand.
 - Analyze and compare the impact of different time quantum values on Round Robin performance, highlighting trade-offs between fairness and efficiency.
 - Ensure fair allocation of CPU resources, minimizing process starvation and optimizing average waiting time.
 - Provide a robust and error-handling mechanism, ensuring that invalid inputs are rejected and user experience remains seamless.
 - Offer insights into real-world applications of Multi-Level Queue Scheduling, particularly in operating systems handling multiple process categories, such as interactive and batch processing.
 - Enhance the educational value of the project by making CPU scheduling concepts more accessible to students and researchers through interactive demonstrations.

Introduction

- In modern computing, efficient CPU scheduling is one of the most critical aspects of operating system design. Scheduling policies impact system responsiveness, throughput, and resource utilization. Multi-Level Queue Scheduling (MLQ) is a widely used scheduling technique that categorizes processes based on priority levels and assigns different scheduling algorithms to each queue.
- Importance of Scheduling Algorithms
 - CPU scheduling plays a significant role in multitasking environments by deciding the order in which processes execute. The effectiveness of scheduling directly influences:
 - System Performance: Proper scheduling reduces waiting time and improves CPU utilization.
 - Process Fairness: Ensures that no process is left starving while maximizing system efficiency.
 - User Experience: Optimized scheduling ensures better responsiveness and load balancing.

Overview

There are various CPU scheduling algorithms used in different operating systems:

- First Come, First Serve (FCFS): Simple but inefficient in preventing long waiting times due to the convoy effect.
- Shortest Job Next (SJN): Prioritizes smaller jobs but may lead to starvation of longer tasks.
- Round Robin (RR): Ensures fairness through time-sharing but may not be efficient for high-priority tasks.
- Multi-Level Queue (MLQ): Segregates processes into queues and applies different scheduling strategies to different priority levels.

Why Multi-Level Queue Scheduling?

- MLQ scheduling is ideal for systems that handle processes with distinct priority levels. This project implements an MLQ scheduler where:
 - High-priority processes execute using Round Robin (RR) for better responsiveness.
 - Low-priority processes execute using First-Come, First-Serve (FCFS) for simple, sequential execution.
- By integrating these techniques, we ensure fairness in execution while optimizing CPU utilization. This project demonstrates how MLQ scheduling can improve process execution and resource allocation, making it a valuable approach in modern operating systems.

3.1 Problem Description

- Efficient CPU scheduling is crucial for optimizing resource utilization while ensuring fair process execution. Traditional scheduling algorithms, such as Round Robin (RR) and First Come, First Serve (FCFS), each have their advantages and limitations:
 - Round Robin (RR) ensures fairness by allocating time slices to processes in a cyclic manner but does not account for priority differences.
 - First Come, First Serve (FCFS) executes processes in the order they arrive, which can lead to inefficiencies in time-sharing and increased waiting time for shorter processes.
- To address these limitations, the Multi-Level Queue Scheduling algorithm is implemented, which categorizes processes into two queues based on priority:
 - Queue 1 (High Priority): Uses Round Robin (RR) to ensure time-sharing fairness among important processes.
 - Queue 2 (Low Priority): Uses First Come, First Serve (FCFS) to process tasks sequentially, reducing overhead for less critical operations.

3.2 Software Requirements

- The Multi-Level Queue Scheduling project is implemented using the following software tools:
 - Python 3.x: The primary programming language used to develop the project due to its simplicity, readability, and extensive library support for computational tasks.
 - Tkinter: A built-in Python GUI framework used to create an interactive and user-friendly interface for adding processes, setting parameters, and visualizing scheduling results.
 - Matplotlib: Utilized for graphical representation of the scheduling process. It helps in generating a Gantt chart, which visually depicts the execution order and time allocation of different processes.

NumPy: A powerful numerical computing library that facilitates efficient array operations, statistical computations, and priority-based sorting of processes for scheduling.

3.3 Hardware Requirements

- To ensure optimal performance and smooth execution of the Multi-Level Queue Scheduling system, the following hardware specifications are recommended:

Processor: Intel Core i3 (or equivalent) and above, ensuring efficient computational performance for process scheduling and visualization tasks.

RAM: A minimum of 4GB is required for basic execution, while 8GB is recommended for seamless processing, especially when handling multiple processes simultaneously.

Storage: At least 200MB of free disk space is needed to accommodate the program files, dependencies, and any generated visualizations.

Operating System: The system is compatible with Windows, Linux, and macOS, ensuring cross-platform usability without major configuration changes.

System Design

4.1 Description of Modules/Programs

The system consists of the following key modules:

- Process Input Module: Allows users to enter arrival time, burst time, and priority values.
- Queue Division Module: Sorts processes based on priority and assigns them to appropriate scheduling queues.
- Scheduling Module: Implements Round Robin (for high-priority processes) and FCFS (for low-priority processes).
- Gantt Chart Module: Displays execution timelines for processes using Matplotlib.
- User Interface Module: Provides a graphical interface for interactive scheduling simulations.
- Performance Evaluation Module: Analyzes process execution time, waiting time, and turnaround time for optimization.
- Error Handling Module: Ensures smooth execution by handling invalid inputs and exceptions.
- Logging and Debugging Module: Maintains logs for debugging scheduling errors.
- Report Generation Module: Generates scheduling performance reports for analysis.

4.2. Methodology

- The Multi-Level Queue (MLQ) Scheduling Algorithm is implemented in this project using a structured and modular approach. The methodology ensures that processes are executed efficiently by classifying them into different queues based on priority and applying appropriate scheduling algorithms. The project follows a step-by-step execution flow, ensuring fairness and optimized CPU utilization. The methodology includes:

- Process Input & Sorting
- Queue Classification & Process Distribution
- Scheduling Execution & CPU Allocation
- Performance Analysis & Metrics Calculation
- Gantt Chart Visualization

- Each of these steps is crucial in managing multiple processes efficiently, ensuring optimal system performance.

Step 1: Process Input & Sorting:

At the beginning of execution, the system prompts users to enter essential details for each process:

- Process ID (PID): Unique identifier for each process.
- Arrival Time (AT): The time at which a process enters the system.
- Burst Time (BT): The total execution time required by the process.
- Priority Level: Defines the importance of the process.

Sorting Mechanism:

- After collecting input, processes are sorted based on their arrival time (AT).
- Sorting is essential to ensure that processes are handled in the correct order.

Step 2: Queue Classification & Process Distribution

Once the processes are sorted, the system classifies them into two separate queues:

→ Queue 1 (High Priority): Uses Round Robin (RR) Scheduling.

→ Queue 2 (Low Priority): Uses First-Come, First-Serve (FCFS) Scheduling.

- Classification Criteria:

→ The median priority value is determined dynamically.

→ Processes with a priority value greater than the median are assigned to Queue1.

→ The remaining processes are assigned to Queue 2.

→ This classification ensures that important tasks receive immediate CPU access, while less critical tasks are handled sequentially.

- Real-World Analogy of Queue Classification

A real-world analogy for queue classification is a hospital's emergency system. Critical patients (high-priority) are treated first using a Round Robin approach, ensuring fairness among emergency cases, while general patients (low-priority) are treated in FCFS order. Similarly, in operating systems, time-sensitive processes (such as real-time user interactions) are scheduled first, while background processes (such as file downloads) are handled sequentially. This ensures that urgent tasks receive immediate CPU access while maintaining efficiency for lower-priority processes.

Step 3: Scheduling Execution & CPU Allocation

Queue 1: High-Priority Execution using Round Robin (RR)

→ The Round Robin (RR) algorithm is applied to Queue 1.

→ A time quantum (TQ) (e.g., 4ms) is predefined.

→ Each process gets executed for a maximum of TQ before moving to the next process.

→ If a process requires more time than TQ, it is moved to the end of Queue 1 for the next cycle.

→ This ensures that no process is left waiting indefinitely.

Queue 2: Low-Priority Execution using FCFS

→ Once Queue 1 is empty, the CPU moves to Queue 2.

→ Processes are executed sequentially based on their arrival time.

→ Once a process starts, it runs to completion without preemption.

→ This approach ensures that urgent tasks are handled first, while non-urgent tasks follow in order

Step 4: Performance Analysis & Metrics Calculation

After execution, the system computes key scheduling metrics to analyze efficiency:

1. Turnaround Time (TAT):

$TAT = \text{Completion Time} - \text{Arrival Time}$

Measures the total duration from process arrival to completion.

2. Waiting Time (WT):

$WT = \text{Turnaround Time} - \text{Burst Time}$

Represents the idle time a process spends before execution.

3. Response Time (RT):

$RT = \text{First Execution Time} - \text{Arrival Time}$

Indicates how quickly a process gets CPU access after arrival.

The system also computes the average for these values to assess overall system efficiency.

Step 5: Gantt Chart Visualization

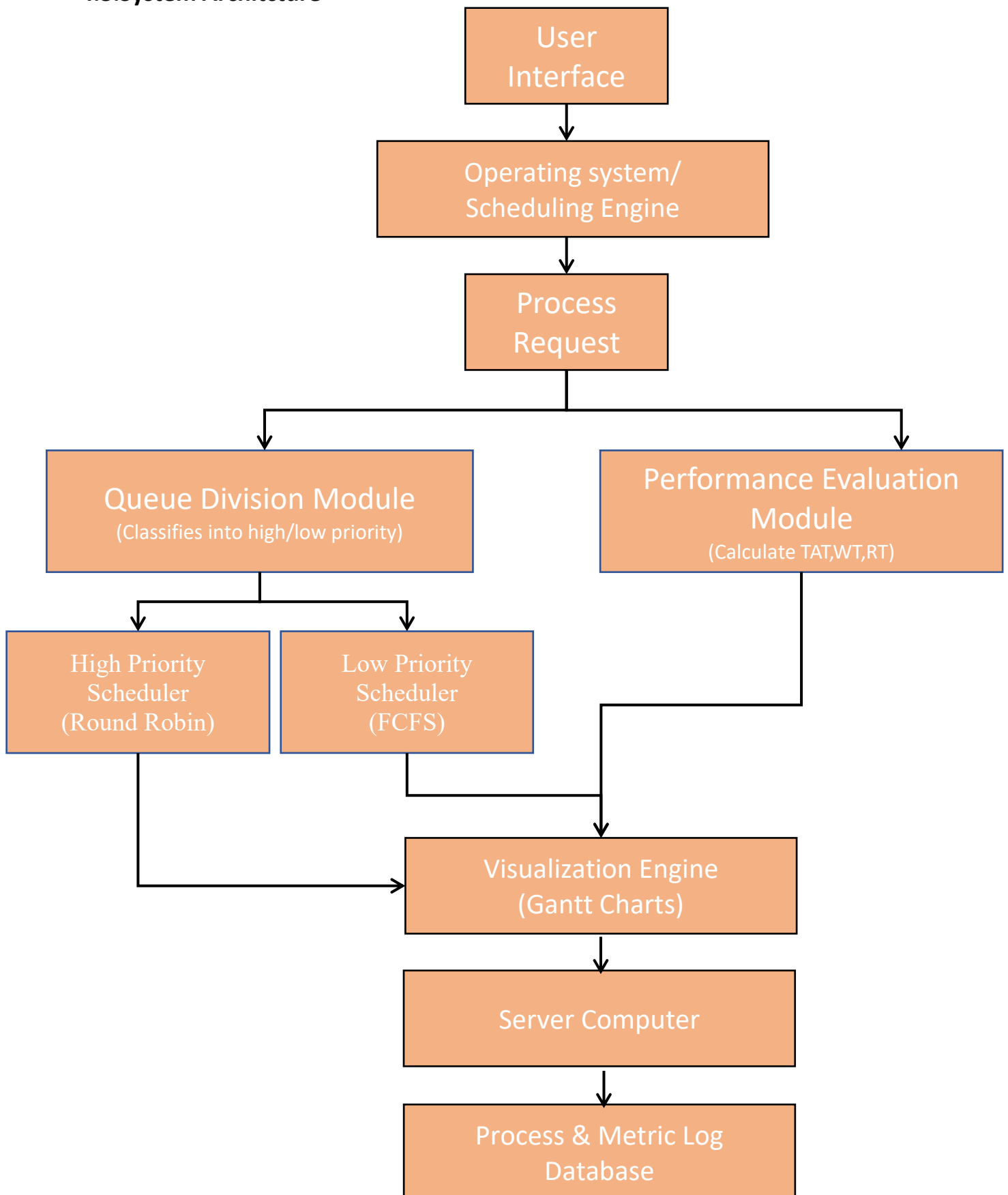
→ A Gantt Chart is generated using Matplotlib to provide a visual representation of process execution:

→ It illustrates start and end times of each process.

→ The time slots for each process are displayed, showing how CPU execution was distributed.

- Helps in understanding CPU utilization and execution fairness.

4.3. System Architecture



Implementation

5.1 Source Code

```
import tkinter as tk
from tkinter import messagebox, ttk
import matplotlib.pyplot as plt
import numpy as np
import random

# Global Variables
process_list = [] # Initialize process_list as empty
time_quantum = 4 # Round Robin time quantum
gantt_box_color = (0.6, 0.8, 1) # Light blue color for all Gantt chart boxes

class Process:
    def __init__(self, pid, arrival_time, burst_time, priority):
        self.pid = f"P{pid}"
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.priority = priority
        self.remaining_time = burst_time
        self.start_time = 0

# Function to Add Process
def add_process():
    try:
        pid = len(process_list) + 1 # Auto-increment Process ID
        at = int(arrival_time_entry.get())
        bt = int(burst_time_entry.get())
        pr = int(priority_entry.get()) # Allow any number for priority

        process_list.append(Process(pid, at, bt, pr))

        # Insert process details into table
        process_table.insert("", tk.END, values=(f"P{pid}", at, bt, pr))

        arrival_time_entry.delete(0, tk.END)
        burst_time_entry.delete(0, tk.END)
        priority_entry.delete(0, tk.END)

        # Make the input boxes smaller after adding a process
        arrival_time_entry.config(width=15) # Decrease the width to make it smaller
        burst_time_entry.config(width=15) # Decrease the width to make it smaller
        priority_entry.config(width=15) # Decrease the width to make it smaller

    except ValueError:
        messagebox.showerror("Input Error", "Enter valid numeric values!")
```



```

# Function to Schedule Processes
def schedule_processes():
    if not process_list:
        messagebox.showwarning("No Processes", "Please add processes before scheduling.")
        return

    process_list.sort(key=lambda p: p.arrival_time) # Sort by arrival time

    # Determine the median priority to split the queues
    priorities = [p.priority for p in process_list]
    median_priority = np.median(priorities)

    queue1 = [p for p in process_list if p.priority >= median_priority] # Higher or equal priority in RR
    queue2 = [p for p in process_list if p.priority < median_priority] # Lower priority in FCFS

    gantt_chart = [] # Initialize gantt_chart
    time = 0

    # Execute Round Robin (Queue 1)
    while queue1:
        process = queue1.pop(0)
        if process.remaining_time == process.burst_time:
            process.start_time = time
            exec_time = min(process.remaining_time, time_quantum)
            process.remaining_time -= exec_time
            gantt_chart.append((process.pid, time, time + exec_time))
            time += exec_time

        if process.remaining_time > 0:
            queue1.append(process)

    # Execute FCFS (Queue 2)
    for process in queue2:
        if process.remaining_time == process.burst_time:
            process.start_time = time
            gantt_chart.append((process.pid, time, time + process.burst_time))
            time += process.burst_time

    show_gantt_chart(gantt_chart)

# Function to Display Gantt Chart
def show_gantt_chart(gantt_chart):
    fig, ax = plt.subplots(figsize=(10, 2))

    for pid, start, end in gantt_chart:
        # Use a single color (light blue) for all processes in the Gantt chart
        ax.broken_barh([(start, end - start)], (5, 5), facecolors=gantt_box_color, edgecolor='black') # Single
        color with black border

```

```
ax.text(start + (end - start) / 2, 7.5, pid, ha='center', va='center', color='black', fontsize=10, fontweight='bold')
```

```
# Set x-axis to only show completion times
ct_values = sorted(list(set([end for _, _, end in gantt_chart])))
ax.set_xticks(ct_values)
ax.set_xticklabels(ct_values)
```

```
ax.set_yticks([]) # Remove y-axis labels
ax.set_xlabel("Completion Time (CT)")
ax.set_title("Multi-Level Queue Scheduling - Gantt Chart")
```

```
plt.show()
```

```
# GUI Design
root = tk.Tk()
root.title("Multi-Level Queue Scheduling")
root.geometry("500x500")
```

```
# Input Fields
tk.Label(root, text="Arrival Time:").pack()
arrival_time_entry = tk.Entry(root, width=15) # Set initial width for input fields
arrival_time_entry.pack()
```

```
tk.Label(root, text="Burst Time:").pack()
burst_time_entry = tk.Entry(root, width=15) # Set initial width for input fields
burst_time_entry.pack()
```

```
tk.Label(root, text="Priority:").pack()
priority_entry = tk.Entry(root, width=15) # Set initial width for input fields
priority_entry.pack()
```

```
# Add Process Button with Color
add_button = tk.Button(root, text="Add Process", command=add_process, bg="#66cc66") # Light green color
add_button.pack()
```

```
# Table to Display Processes
process_table = ttk.Treeview(root, columns=("PID", "AT", "BT", "Priority"), show="headings")
```

```
# Adjust column widths to fit content dynamically based on header text
process_table.column("PID", anchor="center", width=50) # Set a fixed width for PID
process_table.column("AT", anchor="center", width=100) # Adjust width for Arrival Time
process_table.column("BT", anchor="center", width=100) # Adjust width for Burst Time
process_table.column("Priority", anchor="center", width=100) # Adjust width for Priority
```

```
# Set column headers
process_table.heading("PID", text="Process ID")
```

```
process_table.heading("AT", text="Arrival Time")
process_table.heading("BT", text="Burst Time")
process_table.heading("Priority", text="Priority")
```

```
process_table.pack()
```

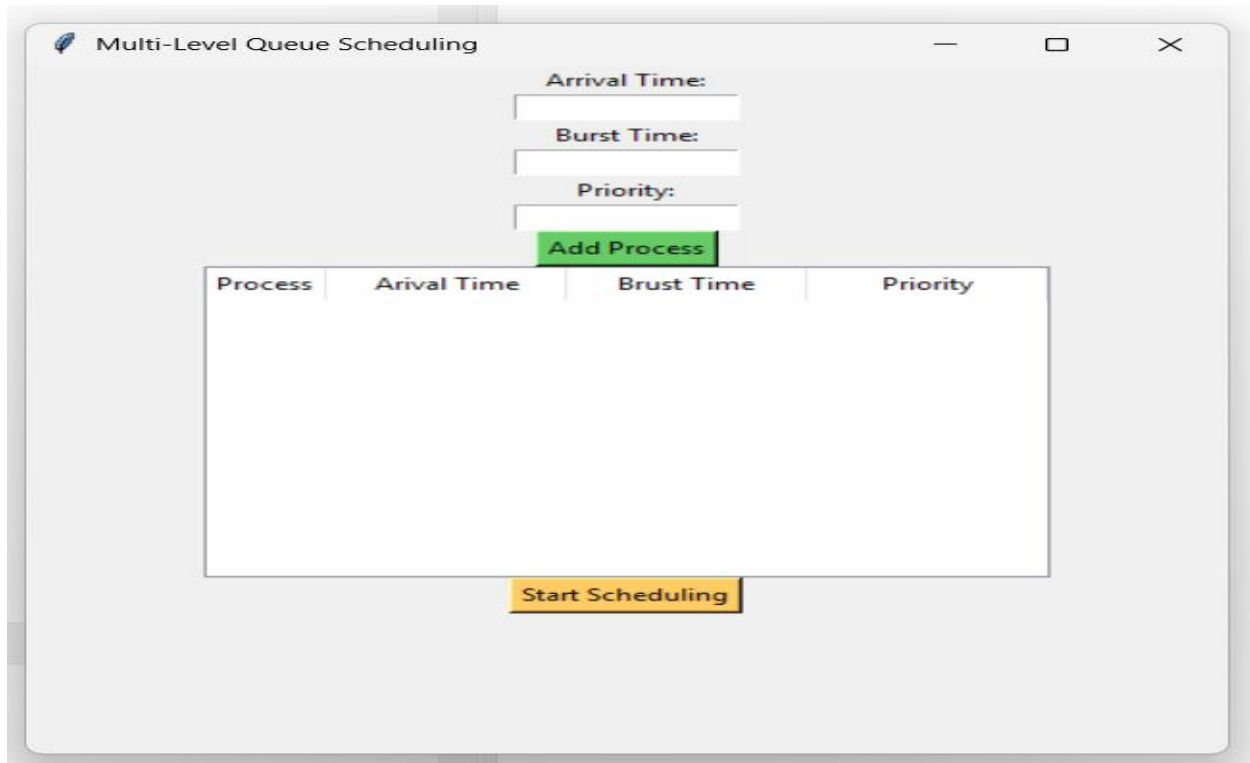
```
# Schedule Button with Color
```

```
schedule_button = tk.Button(root, text="Start Scheduling", command=schedule_processes,  
bg="#ffcc66") # Light yellow color  
schedule_button.pack()
```

```
root.mainloop()
```

5.2.Execution

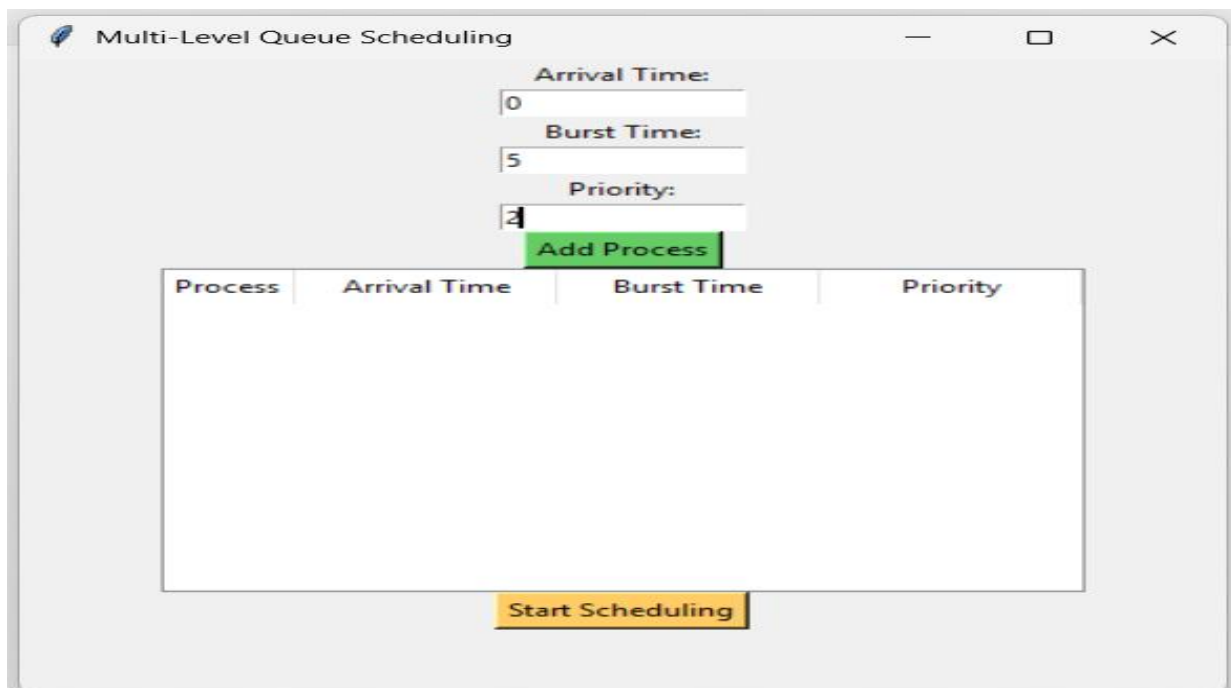
1.Initial UI of Multi-Level Queue Scheduling with empty input fields and process table.



The image shows a window titled "Multi-Level Queue Scheduling". It contains three input fields labeled "Arrival Time:", "Burst Time:", and "Priority:". Below these fields is a green button labeled "Add Process". At the bottom of the window is a yellow button labeled "Start Scheduling". In the center, there is a table with four columns: "Process", "Arival Time", "Brust Time", and "Priority". The table is currently empty.

Process	Arival Time	Brust Time	Priority
---------	-------------	------------	----------

2.A new process with arrival time , burst time , and priority being entered



The image shows the same window as before, but now the input fields contain values: "Arrival Time:" is 0, "Burst Time:" is 5, and "Priority:" is 4. The "Add Process" button is still green, and the "Start Scheduling" button is still yellow. The table remains empty.

Process	Arrival Time	Burst Time	Priority
---------	--------------	------------	----------

3.The process is added to the table after clicking "Add Process."

The screenshot shows a window titled "Multi-Level Queue Scheduling". At the top, there are three input fields labeled "Arrival Time:", "Burst Time:", and "Priority:". Below these fields is a green button labeled "Add Process". Underneath the button is a table with four columns: "Process", "Arrival Time", "Burst Time", and "Priority". The table contains one row of data: P1, 0, 5, 2. At the bottom of the window is a yellow button labeled "Start Scheduling".

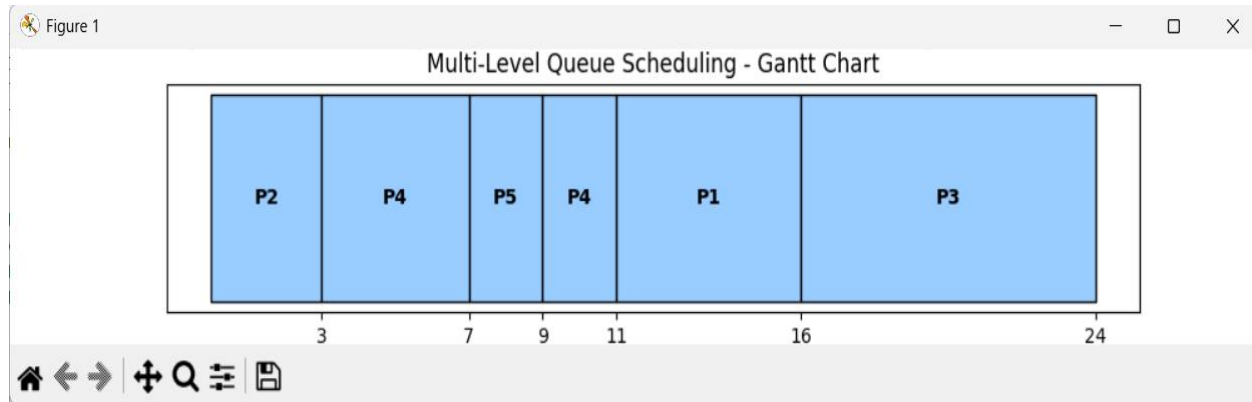
Process	Arrival Time	Burst Time	Priority
P1	0	5	2

4.Process list populated with processes and their attribute

The screenshot shows the same "Multi-Level Queue Scheduling" window. The "Add Process" button is still highlighted in green. The table now contains five rows of data: P1, P2, P3, P4, and P5. At the bottom, the "Start Scheduling" button remains visible.

Process	Arrival Time	Burst Time	Priority
P1	0	5	2
P2	1	3	5
P3	2	8	1
P4	3	6	3
P5	4	2	4

5. Gantt Chart displaying the scheduling order of processes.



Conclusion and future Directions

- The implementation of Multi-Level Queue Scheduling (MLQ) in this project demonstrates the efficient management of processes based on their priority levels. By combining Round Robin (RR) for high-priority tasks and First-Come, First-Serve (FCFS) for low-priority tasks, we achieved a balance between fairness and efficiency in CPU scheduling.
- The graphical user interface (GUI) developed using Tkinter allows users to input process details, visualize execution order, and analyze scheduling performance through a Gantt chart. This project successfully highlights how multi-level scheduling is utilized in modern operating systems to optimize resource allocation and improve response time.
- Key takeaways from this project include:
 - Fair process execution by dynamically assigning scheduling algorithms.
 - Improved CPU utilization by reducing waiting time and prioritizing tasks efficiently.
 - Better user interaction through a simple and interactive Python-based GUI.
 - Visualization of scheduling using Gantt charts for a clear understanding of execution flow.
- Real-World Applications of Multi-Level Queue Scheduling:
 - Real-Time & Embedded Systems:

MLQ is crucial in real-time systems such as autonomous vehicles, medical devices, and industrial automation. In a self-driving car, for example:

 - High-priority queue: Critical tasks like sensor data processing and obstacle detection are scheduled with a fast, responsive algorithm (Round Robin).
 - Low-priority queue: Non-urgent tasks like logging and system diagnostics are handled using FCFS.
 - Operating Systems: Used in Windows, Linux, and macOS to manage system processes, separating interactive tasks (foreground) from background processes.

- Future Directions:

To further improve this scheduling model, the following enhancements can be considered:

→ Dynamic Time Quantum Adjustment – Instead of a fixed Round Robin time slice, implementing a variable quantum based on system load can improve efficiency.

→ Priority Aging Mechanism – To prevent starvation, a mechanism to increase priority over time can be introduced.

→ Multi-Processor Scheduling – Extending this model to support multi-core CPU scheduling for real-world applications.

→ More Scheduling Policies – Adding support for Shortest Job Next (SJN) or Priority Preemptive Scheduling to compare different scheduling techniques.

- By incorporating these improvements, the Multi-Level Queue Scheduling algorithm can be further optimized to enhance its real-world applicability in modern operating systems.

References

Online Tutorials:

<https://www.geeksforgeeks.org/multilevel-queue-scheduling/>

YouTube Tutorials:For MLQ

<https://youtu.be/A8OmWZkYCxg?si=RG02kPs1p9C9Xr6m>

https://youtu.be/JDsrc0LKKhg?si=es_oCr4Rku-mONqg

For Python

https://youtube.com/playlist?list=PLS1QulWo1RIY6fmY_itjEhCMsdtAjgbZM&si=ITUCqP_VY3hUvzWP

<https://youtube.com/playlist?list=PLC2mgeYbYNm-1V5A55ZVVtyAGUuTnEoZm&si=zRXjp-MTkzxz3YFA>

Books:

Operating System Concepts: <https://os-book.com/>