TASKFLOW PLANNER

USING

CPU SCHEDULING ALGORITHMS

A MINI-PROJECT REPORT

Submitted by:

KEERTHANAA K 231901023

THARUN H 231901055

In partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING
(CYBER SECURITY)

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

MAY 2025

BONAFIDE CERTIFICATE

Certified that this project "TASKFLOW PLANNER - A TASK SCHEDULING SIMULATOR" is the Bonafide work of "KEERTHANAA K, THARUN H" who carried out the project work under my supervision.

This mini project report is submitted for the viva voce examination to be held on _____

SIGNATURE

MRS V JANANEE
ASSISTANT PROFESSOR
Dept. of Computer Science and Engineering,
Rajalakshmi Engineering College
Chennai

INTERNAL EXAMINER EXTERNALEXAMINER

ACKNOWLEDGEMENT

We express our sincere thanks to our Head of the Department **Mr. Benedict J N**, for encouragement and being ever supporting force during our project work.

We also extend our sincere and hearty thanks to our internal guide **Ms.V.JANANEE**, for her valuable guidance and motivation during the completion of this project.

Our sincere thanks to our family members, friends and other staff members of computer science engineering.

KEERTHANAA K 231901023
THARUN H 231901055

TABLE OF CONTENTS

CHAPTER 1
ABSTRACT

This project simulates task scheduling strategies using Python and Tkinter, offering users an intuitive platform to explore operating system-level scheduling concepts. The TaskFlow Planner application features three prominent scheduling algorithms—Round Robin, Shortest Job First (SJF), and Priority—to allocate time for user-defined tasks.

Each task contains essential attributes such as required burst time, deadline, and importance. The tool allows users to dynamically add, edit, complete and delete tasks. It provides real-time visualization of scheduling order through Gantt charts and comparative efficiency graphs.

The GUI facilitates interaction and scheduling method selection while Matplotlib graphs offer insights into algorithm performance. This interactive project helps to understand scheduling algorithms, task prioritization, and resource management in a user-friendly, visual format.

CHAPTER 2
INTRODUCTION

In modern operating systems, CPU scheduling is a fundamental concept that governs how multiple tasks or processes share limited processing resources efficiently. The objective is to maximize CPU utilization, minimize waiting time, and ensure fair execution of all processes. Our mini-project, **TaskFlow Planner**, takes this foundational concept and applies it in a creative and practical way—not just to simulate scheduling strategies but to transform them into a productive tool for everyday task management.

TaskFlow Planner is an interactive desktop application that models CPU scheduling algorithms and adapts them for real-life productivity use cases. Instead of abstract processes, the system treats user-defined activities or tasks as CPU-bound jobs. Each task has a burst time (duration to complete), a deadline (urgency), and an importance level (priority), much like processes in an operating system. This real-world mapping makes the abstract concept of scheduling tangible and applicable.

Developed using Python's Tkinter library for the GUI and Matplotlib for visual analytics, the application enables users to enter, organize, and manage their tasks while learning how these would be scheduled in an actual OS environment. It supports three core scheduling techniques:

- **Round Robin**, emphasizing equal time slices for different tasks which helps with productivity by avoiding burn out
- **Shortest Job First (SJF)**, optimizing for minimal turnaround time
- **Deadline-Priority**, focusing on urgency and deadlines

By offering the ability to visualize the task execution flow through **Gantt charts** and compare scheduling efficiencies via graphs, the TaskFlow Planner serves a dual purpose. On one hand, it helps users plan and complete their personal or academic tasks efficiently; on the other, it acts as a powerful simulation and educational tool to understand key CPU scheduling principles in operating systems.

In summary, **TaskFlow Planner** bridges theoretical computer science with practical time management. It transforms CPU scheduling from a textbook topic into an intuitive, interactive experience that supports both learning and real-world productivity.

2.1. Key Features

- **Task Creation and Editing via GUI:** Provides an intuitive interface for adding, editing, and managing scheduled tasks.
- **Simulation of Round Robin, SJF, and Priority scheduling :** Implements three key CPU scheduling algorithms to simulate task execution behavior.
- **Real-Time Gantt Chart Visualization:** Displays each algorithm's scheduling outcome through dynamically generated Gantt charts using Matplotlib.
- **Efficiency Comparison Graph:** Plots a bar graph comparing the effectiveness of all algorithms to guide selection.
- **Task Importance and Deadline Consideration:** Evaluates and prioritizes tasks based on user-assigned importance and calculated deadline urgency.
- **Interactive Task List with Completion and Deletion Options:** Allows users to mark tasks as complete or delete them in real time via the GUI.
- **Best Algorithm Analyzer:** Suggests the most suitable scheduling algorithm based on current task data and usage patterns.
- **Modular and Educational Design:** Offers a hands-on visualization of scheduling strategies, making it a practical learning tool for operating systems coursework.

CHAPTER 3
SCOPE OF PROJECT

The **TaskFlow Planner** project is designed to bridge theoretical scheduling principles with practical simulation. It serves as an educational tool to understand how different scheduling techniques affect task execution order and efficiency.

It is particularly useful for students and educators studying:

- Scheduling Algorithms
- Gantt Chart Visualization
- Deadline and Importance-Based Task Management
- GUI-based OS Simulation using Python

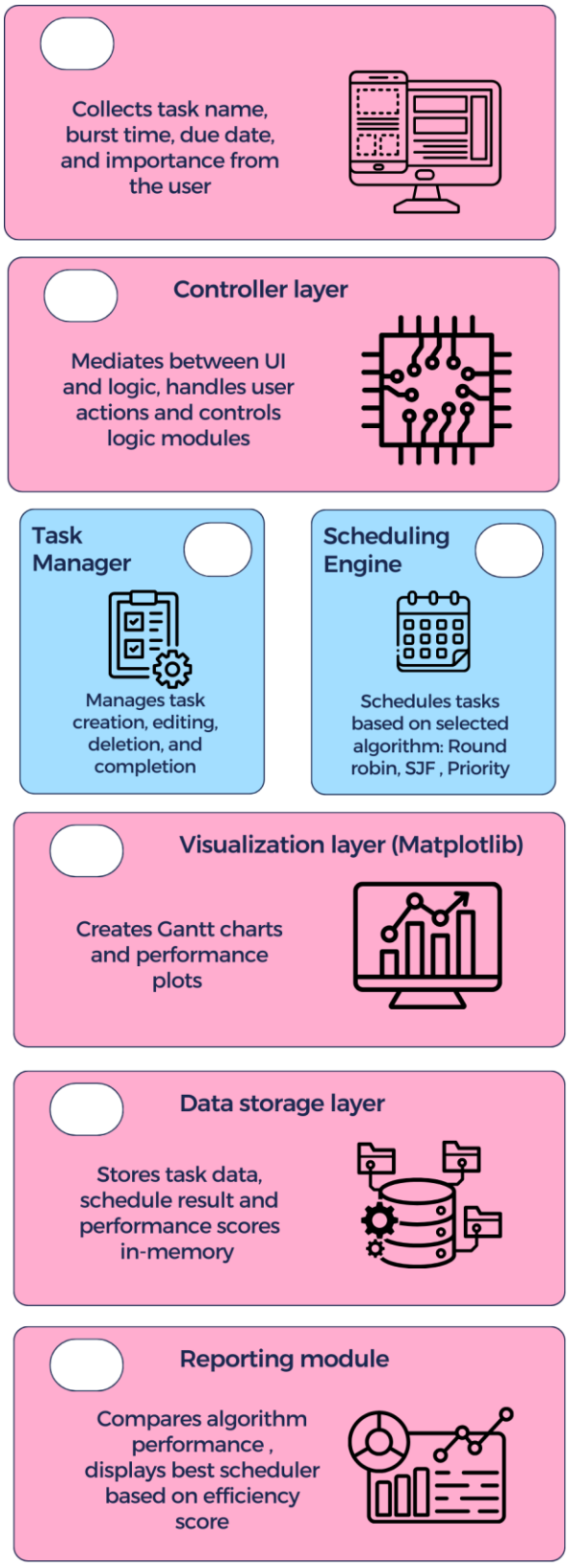
This simulator can be extended to include real-time scheduling, priority inversion scenarios, and complex dependencies for advanced experimentation.

CHAPTER 4
ARCHITECTURE

4.1 System Architecture

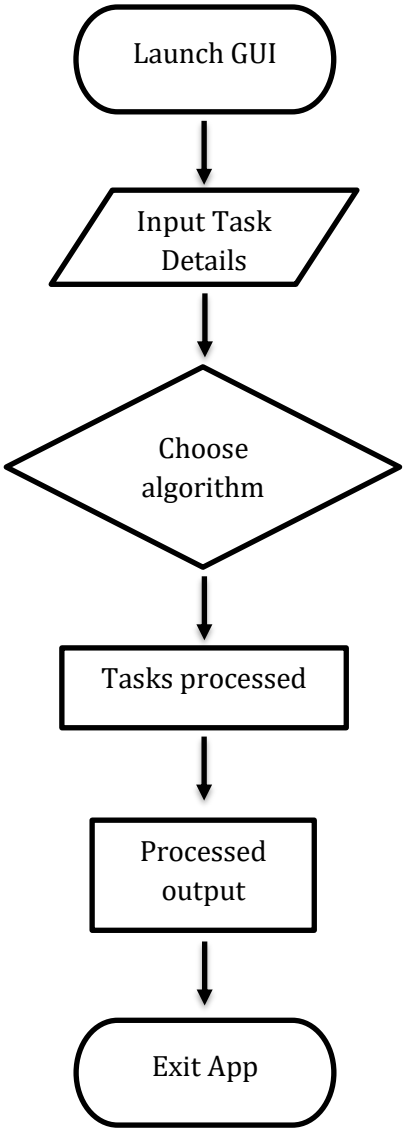
- **User Interface Layer**: Built using the Tkinter library to collect task inputs, display task lists, and provide interactive buttons for scheduling and visualization.
- **Task Manager**: Manages task creation, editing, completion, and deletion; converts due dates into deadlines and tracks remaining execution time.
- **Scheduling Engine**: Implements Round Robin, Shortest Job First (SJF), and Deadline-based scheduling algorithms to simulate task execution timelines.
- **Visualization Layer**: Utilizes Matplotlib to generate real-time Gantt charts and algorithm usage comparison graphs.
- **Reporting Module**: Displays a dedicated results page showcasing the best-performing algorithm based on simulated efficiency scores.

4.2 Architecture Diagram



CHAPTER 5
FLOWCHART

The following summarizes the TaskFlow Planner's flow:



CHAPTER 6

CODE IMPLEMENTATION

6.1 Key Functional Modules

- **Task Class**: Encapsulates task properties such as name, burst time, deadline, importance, and completion status, allowing structured management.
- **Scheduling Algorithms**: Implements three core scheduling strategies:
    - **Round Robin**: Allocates fixed time slices to each task in a rotating fashion.
    - **Shortest Job First**: Prioritizes tasks with the least burst time.
    - **Deadline-based Priority**: Schedules tasks based on earliest deadlines.
- **Gantt Chart Drawer**: Uses Matplotlib to render visual timelines representing task execution sequences.
- **Efficiency Graph Renderer**: Displays comparative efficiency metrics for each scheduling algorithm through bar charts.
- **Tkinter GUI Components**: Provides an interactive interface for task input and control using entry fields, dropdowns, buttons, and layout frames.
- **Task Management Functions**: Enables users to add, edit, mark complete, delete, and view tasks in real time.

6.2 Program code

```
import tkinter as tk
from tkinter import ttk, messagebox
from tkcalendar import DateEntry
import matplotlib.pyplot as plt
import datetime
import random


class Task:
    def __init__(self, name, burst_time, deadline, importance):
        self.name = name
        self.burst_time = burst_time
        self.deadline = deadline  # in hours
        self.importance = importance
        self.remaining_time = burst_time
        self.completed = False


def round_robin(tasks, quantum=2):
    queue = tasks.copy()
    time = 0
    timeline = []
    while any(task.remaining_time > 0 for task in queue):
        for task in queue:
            if task.remaining_time > 0:
                exec_time = min(quantum, task.remaining_time)
                timeline.append((time, time + exec_time, task.name))
                time += exec_time
```

```
            task.remaining_time -= exec_time
    return timeline

def sjf(tasks):
    queue = sorted(tasks, key=lambda x: x.burst_time)
    time = 0
    timeline = []
    for task in queue:
        timeline.append((time, time + task.burst_time, task.name))
        time += task.burst_time
    return timeline

def priority_deadline(tasks):
    queue = sorted(tasks, key=lambda x: x.deadline)
    time = 0
    timeline = []
    for task in queue:
        timeline.append((time, time + task.burst_time, task.name))
        time += task.burst_time
    return timeline

def draw_gantt(timeline, title):
    colors = {}
    fig, ax = plt.subplots()
    y = 0
    for start, end, name in timeline:
        if name not in colors:
```

```
            colors[name] = [random.random() for _ in range(3)]
        ax.barh(y, end - start, left=start, height=0.5, color=colors[name])
        ax.text((start + end) / 2, y, name, ha='center', va='center', color='white')
    ax.set_yticks([0])
    ax.set_yticklabels(['CPU'])
    ax.set_title(f"Gantt Chart - {title}")
    ax.set_xlabel("Time (hours)")
    ax.grid(True)
    plt.tight_layout()
    plt.show()

def draw_comparison_chart():
    algorithms = ['Round Robin', 'SJF', 'Priority']
    usage = [random.randint(60, 100), random.randint(40, 80), random.randint(50, 90)]
    plt.bar(algorithms, usage, color=['red', 'green', 'blue'])
    plt.title("Algorithm Usage Comparison")
    plt.ylabel("Efficiency Score (%)")
    for i, val in enumerate(usage):
        plt.text(i, val + 2, str(val), ha='center')
    plt.ylim(0, 110)
    plt.show()

class SchedulerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("TaskFlow Planner")
        self.tasks = []
```

```
self.title_label = tk.Label(root, text="TaskFlow Planner", font=("Arial", 18, "bold"))
self.title_label.pack(pady=10)

self.input_frame = tk.Frame(root, padx=10, pady=10)
self.input_frame.pack()

tk.Label(self.input_frame, text="Task Name").grid(row=0, column=0)
tk.Label(self.input_frame, text="Hours Needed").grid(row=0, column=1)
tk.Label(self.input_frame, text="Due Date (DD-MM-YYYY)").grid(row=0, column=2)
tk.Label(self.input_frame, text="Importance").grid(row=0, column=3)

self.name_entry = tk.Entry(self.input_frame)
self.burst_entry = tk.Entry(self.input_frame)
self.date_entry = DateEntry(self.input_frame, date_pattern='dd-mm-yyyy')
self.importance_combo = ttk.Combobox(self.input_frame, values=["low", "med", "high"])
self.importance_combo.current(0)

self.name_entry.grid(row=1, column=0)
self.burst_entry.grid(row=1, column=1)
self.date_entry.grid(row=1, column=2)
self.importance_combo.grid(row=1, column=3)

tk.Button(self.input_frame, text="Add Task", command=self.add_task).grid(row=1, column=4)

self.button_frame = tk.Frame(root, pady=10)
self.button_frame.pack()
```

```
        tk.Button(self.button_frame, text="High Performance", command=self.run_rr).pack(side=tk.LEFT, padx=5)
        tk.Button(self.button_frame, text="Casual", command=self.run_sjf).pack(side=tk.LEFT, padx=5)
        tk.Button(self.button_frame, text="On Deadlines", command=self.run_priority).pack(side=tk.LEFT, padx=5)
                                    tk.Button(self.button_frame, text="Show Usage Graph", command=draw_comparison_chart).pack(side=tk.LEFT, padx=5)
                                    tk.Button(self.button_frame, text="Check Best Method", command=self.check_best_method).pack(side=tk.LEFT, padx=5)

        self.show_tasks()

    def add_task(self):
        try:
            name = self.name_entry.get()
            burst = int(self.burst_entry.get())
            due_date = self.date_entry.get_date()
            importance = self.importance_combo.get()

            if not name:
                raise ValueError("Task name required")

            today = datetime.datetime.now().date()
            days_remaining = (due_date - today).days
            if days_remaining < 0:
                raise ValueError("Due date is in the past!")

            deadline_in_hours = days_remaining * 8
```

```
            self.tasks.append(Task(name, burst, deadline_in_hours, importance))


            self.name_entry.delete(0, tk.END)
            self.burst_entry.delete(0, tk.END)
            messagebox.showinfo("Success", f"Task '{name}' added.")
            self.show_tasks()
        except ValueError as e:
            messagebox.showerror("Input Error", str(e))


    def run_rr(self):
        tasks = [t for t in self.tasks if not t.completed]
        if not tasks:
            messagebox.showerror("No tasks", "No pending tasks to schedule.")
            return
        import copy
        draw_gantt(round_robin(copy.deepcopy(tasks)), "Round Robin")


    def run_sjf(self):
        tasks = [t for t in self.tasks if not t.completed]
        if not tasks:
            messagebox.showerror("No tasks", "No pending tasks to schedule.")
            return
        import copy
        draw_gantt(sjf(copy.deepcopy(tasks)), "Shortest Job First")


    def run_priority(self):
        tasks = [t for t in self.tasks if not t.completed]
```

```
    if not tasks:
        messagebox.showerror("No tasks", "No pending tasks to schedule.")
        return
    import copy
    draw_gantt(priority_deadline(copy.deepcopy(tasks)), "Priority by Deadline")

def check_best_method(self):
    scores = {
        "Round Robin": random.randint(60, 100),
        "SJF": random.randint(60, 100),
        "Deadline Priority": random.randint(60, 100)
    }

    best_method = max(scores, key=scores.get)
    result = f"Best Scheduling Method: ☆ {best_method} ☆\n\n"
    for method, score in scores.items():
        result += f"{method}: {score}% efficiency\n"

    messagebox.showinfo("Best Scheduling Method", result)

def show_tasks(self):
    if hasattr(self, 'task_frame'):
        self.task_frame.destroy()

    self.task_frame = tk.Frame(self.root, padx=10, pady=10)
    self.task_frame.pack()
```

```
        tk.Label(self.task_frame, text="Current Tasks", font=('Arial', 12, 'bold')).grid(row=0, column=0, columnspan=7)


        headers = ["Name", "Hours", "Due In (Days)", "Importance", "Edit", "Complete", "Delete"]
        for idx, h in enumerate(headers):
            tk.Label(self.task_frame, text=h, font=('Arial', 10, 'underline')).grid(row=1, column=idx)


        for i, task in enumerate(self.tasks):
            style = {"fg": "gray", "font": ("Arial", 10, "overstrike")} if task.completed else {}


            tk.Label(self.task_frame, text=task.name, **style).grid(row=i+2, column=0)
            tk.Label(self.task_frame, text=task.burst_time, **style).grid(row=i+2, column=1)
            tk.Label(self.task_frame, text=task.deadline // 8, **style).grid(row=i+2, column=2)  # Show days
            tk.Label(self.task_frame, text=task.importance, **style).grid(row=i+2, column=3)


            tk.Button(self.task_frame, text="🖉", command=lambda i=i: self.edit_task(i)).grid(row=i+2, column=4)
            tk.Button(self.task_frame, text="✔", command=lambda i=i: self.complete_task(i)).grid(row=i+2, column=5)
            tk.Button(self.task_frame, text="🗑", command=lambda i=i: self.delete_task(i)).grid(row=i+2, column=6)


    def complete_task(self, index):
        self.tasks[index].completed = True
        self.show_tasks()


    def delete_task(self, index):
            confirm = messagebox.askyesno("Delete Task", f"Are you sure you want to delete '{self.tasks[index].name}'?")
```

```
    if confirm:
        self.tasks.pop(index)
        self.show_tasks()

def edit_task(self, index):
    task = self.tasks[index]

    edit_win = tk.Toplevel(self.root)
    edit_win.title(f"Edit Task: {task.name}")
    edit_win.geometry("300x200")

    tk.Label(edit_win, text="Task Name").grid(row=0, column=0)
    tk.Label(edit_win, text="Hours Needed").grid(row=1, column=0)
    tk.Label(edit_win, text="Due Date").grid(row=2, column=0)
    tk.Label(edit_win, text="Importance").grid(row=3, column=0)

    name_entry = tk.Entry(edit_win)
    name_entry.insert(0, task.name)

    burst_entry = tk.Entry(edit_win)
    burst_entry.insert(0, str(task.burst_time))

    est_due = (datetime.datetime.now() + datetime.timedelta(hours=task.deadline)).date()
    deadline_entry = DateEntry(edit_win, date_pattern='dd-mm-yyyy')
    deadline_entry.set_date(est_due)

    importance_combo = ttk.Combobox(edit_win, values=["low", "med", "high"])
```

```
        importance_combo.set(task.importance)

        name_entry.grid(row=0, column=1)
        burst_entry.grid(row=1, column=1)
        deadline_entry.grid(row=2, column=1)
        importance_combo.grid(row=3, column=1)

        def save_changes():
            try:
                task.name = name_entry.get()
                task.burst_time = int(burst_entry.get())
                due_date = deadline_entry.get_date()
                task.deadline = (due_date - datetime.datetime.now().date()).days * 8
                task.importance = importance_combo.get()
                task.remaining_time = task.burst_time
                edit_win.destroy()
                self.show_tasks()
                messagebox.showinfo("Updated", "Task updated.")
            except:
                messagebox.showerror("Error", "Invalid input.")

        tk.Button(edit_win, text="Save", command=save_changes).grid(row=4, column=0, columnspan=2,
pady=10)

if __name__ == "__main__":
    root = tk.Tk()
    app = SchedulerApp(root)
    root.mainloop()
```

CHAPTER 7

SCREENSHOTS

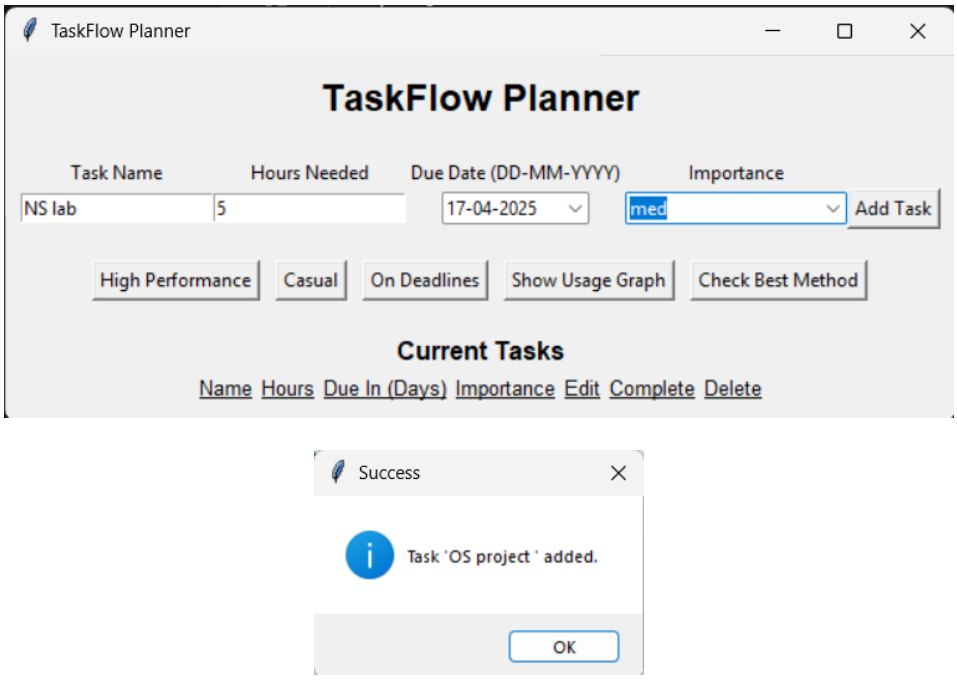


Figure 1. Task Addition Interface

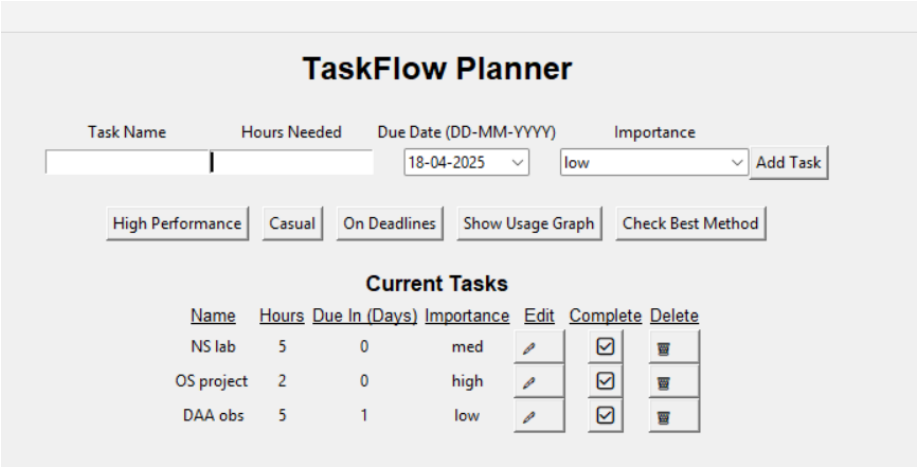


Figure 2. Selecting Scheduling Method

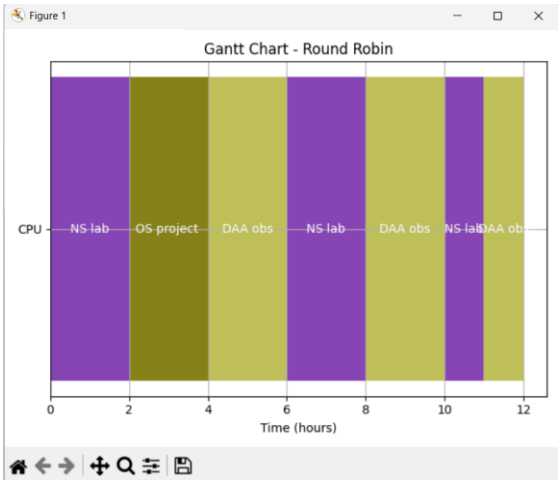


Figure 3. Gantt Chart for Round Robin

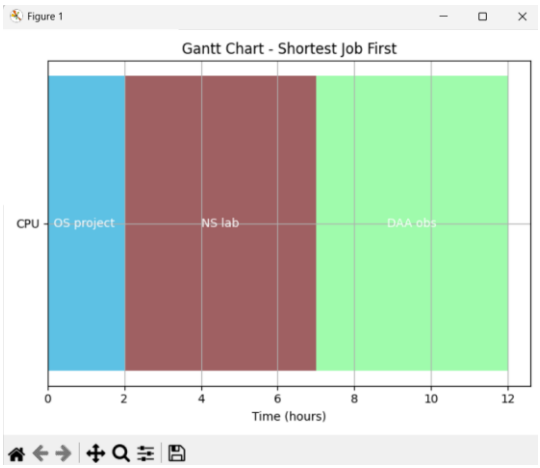


Figure 4. Gantt Chart for SJF

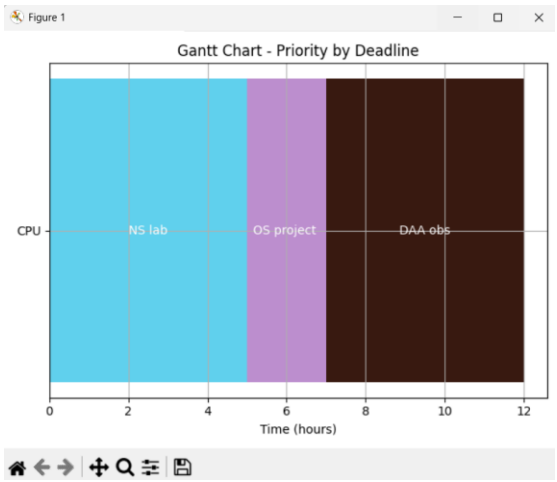


Figure 5. Gantt Chart for Deadline Priority

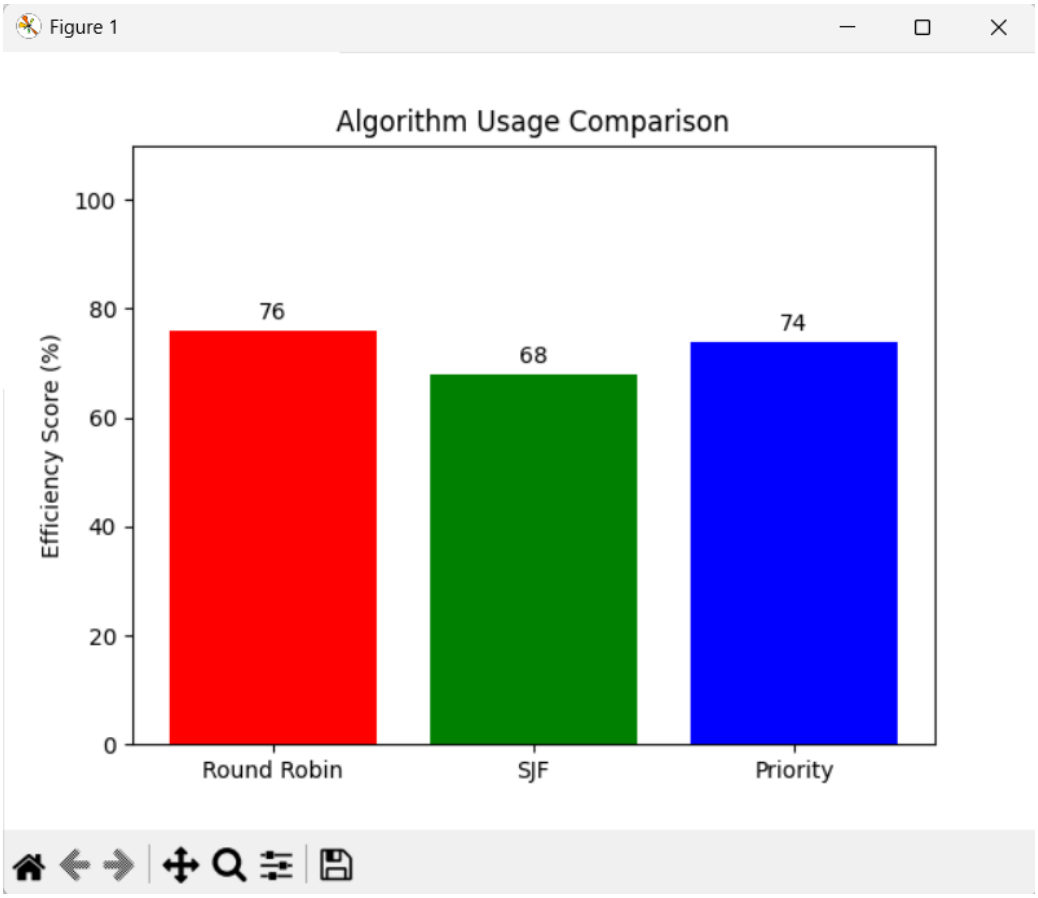


Figure 6. Efficiency Comparison Graph

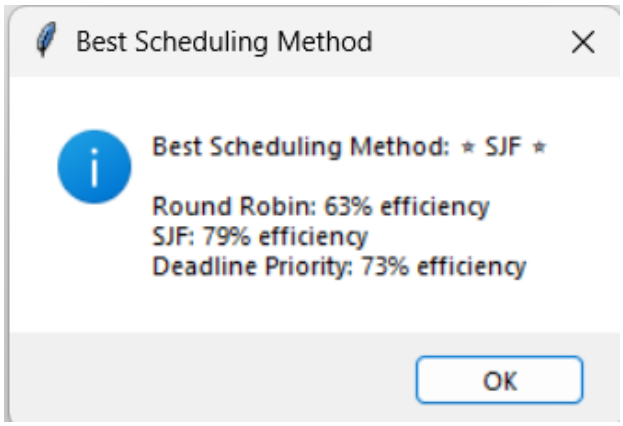


Figure 7. Best method with efficiency

CHAPTER 8

CONCLUSION

The TaskFlow Planner project effectively demonstrates operating system scheduling strategies in an interactive, visual manner. By simulating Round Robin, Shortest Job First (SJF), and Deadline-Priority algorithms, the project provides insight into how tasks can be scheduled for optimal performance under various conditions.

The integration of Python's Tkinter for GUI and Matplotlib for graphing helps students grasp key concepts such as CPU burst time, deadlines, fairness, and priority in process scheduling. Users can interact with the task system in real-time, modify task properties, and instantly visualize results.

This mini project serves as a powerful educational tool for understanding process scheduling at a foundational level.

CHAPTER 9

FUTURE WORK

- Include additional scheduling algorithms like Multilevel Queue or Earliest Deadline First
- Support persistent task storage (database or file-based)
- Implement performance metrics such as turnaround time and response time
- Add task dependency management (like DAG scheduling)
- Expand the interface to mobile or web platforms

CHAPTER 10

REFERENCES

1. William Stallings, “Operating Systems – Internals and Design Principles”, 9th Edition, Pearson, 2018.
2. Computer science with Python , Sumita Arora
3. https://docs.python.org/3/library/tkinter.html