Project of DAA

Job Scheduling Algorithm Report

Subject: Design and Analysis of Algorithms(24CAP-612)

Submitted by: Anuj and Mannat Mahajan(24MCI10020,24MCI10032)

Section\group:24MAM1(A) Branch:MCA(AIML)

Github:- https://github.com/anujdhiman28/Job-Scheduling-Algorithm-.git

https://github.com/mannatmahajan5/daaproject

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

Table of Contents

6. Conclusion

- 6.1 Summary of Achievements
- 6.2 Key Features of the Application
- 6.3 Potential for Future Enhancements

7. Future Framework

- 7.1 Additional Scheduling Algorithms
- 7.2 Save/Load Functionality
- 7.3 Improved Visualization
- 7.4 Error Handling Enhancements

8. Learning Outcomes

- 8.1 Understanding Job Scheduling
- 8.2 Sorting Algorithms
- 8.3 GUI Development
- 8.4 Data Visualization
- 8.5 Programming Skills

Aim:-

The primary aim of this project is to design and implement a job scheduling application that employs the First-Come, First-Served (FCFS) algorithm to process jobs based on their arrival times. By integrating the Merge Sort algorithm for efficient sorting, the application enhances user experience through a graphical user interface (GUI) built using Tkinter. Additionally, the application provides visual feedback on job performance using Matplotlib, making the results easy to interpret.

Objective/Problem Definition:-

The key objectives of this project include:

1. **Job Input**: Allow users to input job details, including job names, arrival times, and burst times.
2. **Sorting Jobs**: Implement a sorting algorithm to order jobs based on their arrival times, ensuring accurate scheduling.
3. **Scheduling Calculation**: Calculate waiting and turnaround times for each job in the queue.
4. **User-Friendly Interface**: Create a GUI that is intuitive and easy to use, enabling users to input data without hassle.
5. **Visualization**: Provide a graphical representation of the results, allowing users to quickly assess job performance metrics.

The project addresses the problem of efficiently managing job execution based on the arrival time and burst time, which is crucial in various computing environments.

Programming Languages Used:-

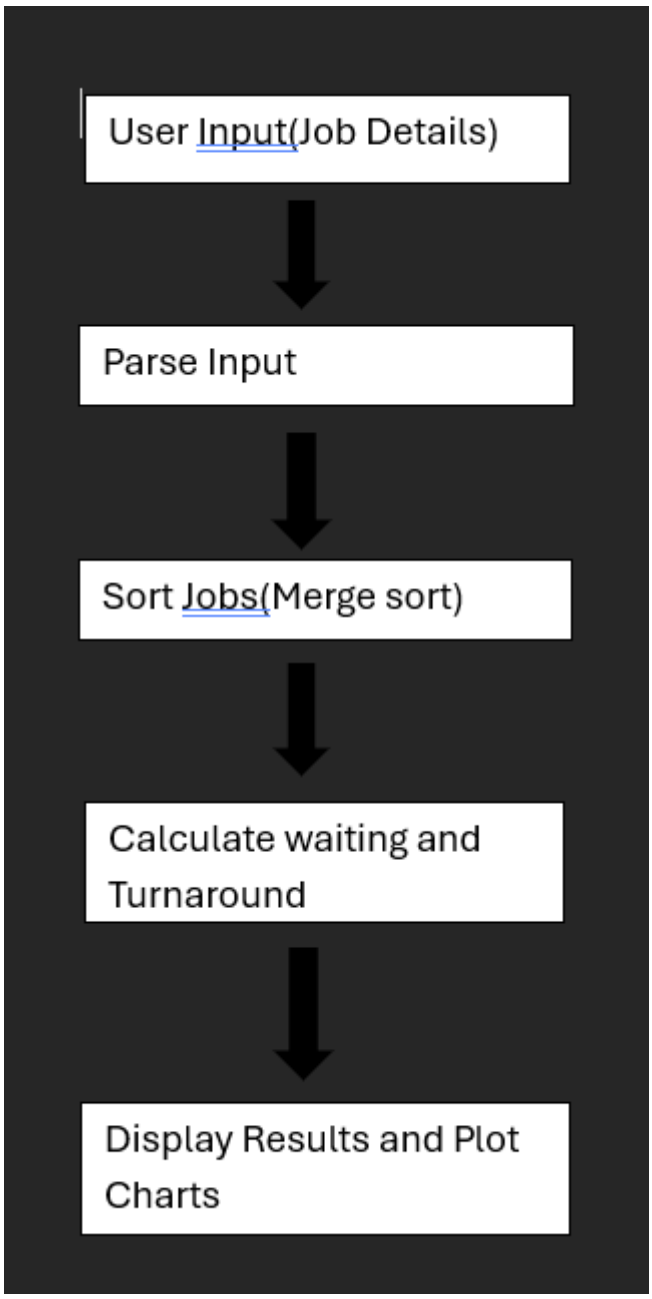
The project utilizes the following programming languages and libraries:

- **Python**: The primary programming language used to develop the application.
- **Tkinter**: A standard GUI toolkit in Python that allows for the creation of user interfaces.
- **Matplotlib**: A plotting library for Python, used for creating static, interactive, and animated visualizations.

Block Diagram/Design Flow/Flow Chart:-

Block Diagram

The block diagram outlines the flow of the application from user input to output:



Flow Chart:-

The flowchart illustrates the sequence of operations in the application:

1. Start
2. Display GUI
3. Get user input for jobs
4. Parse input data
5. Sort jobs using Merge Sort
6. Calculate waiting and turnaround times
7. Display results
8. Plot waiting time chart
9. End

Algorithm or Pseudo Code:-

Pseudo Code

The following pseudo code outlines the major functions of the application:

1. **Define Job Class**:
    - Attributes: name, arrival_time, burst_time.
2. **Function merge_sort(jobs)**:
    - If the length of jobs is greater than 1:
        - Split the list into two halves.
        - Recursively call merge_sort on both halves.
        - Merge the sorted halves back together based on arrival_time.
3. **Function calculate_scheduling()**:
    - Get job input from the text area.
    - Parse the input into job objects.
    - Call merge_sort(jobs) to sort the jobs.
    - Initialize waiting_time to 0.
    - For each job:
        - Calculate turnaround time.

        - Update waiting times.
    - Display the results and call plot_waiting_time_chart().
4. **Function plot_waiting_time_chart(jobs, waiting_times)**:
    - Create a bar chart showing the waiting times for each job.
5. **GUI Setup**:
    - Create the main window and labels.
    - Set up the text area for job input.
    - Add a button to trigger scheduling calculation.
    - Display the results.

Implementation:-

The implementation consists of the following code:

```
import tkinter as tk
from tkinter import messagebox
import matplotlib.pyplot as plt


class Job:
    def __init__(self, name, arrival_time, burst_time):
        self.name = name
        self.arrival_time = arrival_time
        self.burst_time = burst_time


def merge_sort(jobs):
    if len(jobs) > 1:
        mid = len(jobs) // 2
        left_half = jobs[:mid]
        right_half = jobs[mid:]


        merge_sort(left_half)
```

```
        merge_sort(right_half)

        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i].arrival_time < right_half[j].arrival_time:
                jobs[k] = left_half[i]
                i += 1
            else:
                jobs[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            jobs[k] = left_half[i]
            i += 1
            k += 1
        while j < len(right_half):
            jobs[k] = right_half[j]
            j += 1
            k += 1

def calculate_scheduling():
    try:
        jobs = []
        job_input = job_entry.get("1.0", tk.END).strip().split("\n")

        for job in job_input:
            name, arrival, burst = job.split()
```

```
            jobs.append(Job(name, int(arrival), int(burst)))

        merge_sort(jobs)

        waiting_time = 0
        turnaround_time = []
        waiting_times = []
        results = []

        for job in jobs:
            turnaround = waiting_time + job.burst_time
            turnaround_time.append(turnaround)
            waiting_times.append(waiting_time)
            results.append(f"Job {job.name}: Waiting Time = {waiting_time}, Turnaround Time=
            {turnaround}")
            waiting_time += job.burst_time

        result_label.config(text="\n".join(results))
        plot_waiting_time_chart(jobs, waiting_times)
    except Exception:
        messagebox.showerror("Input Error", "Please enter valid job details in the correct
format.")

def plot_waiting_time_chart(jobs, waiting_times):
    job_names = [job.name for job in jobs]

    plt.bar(job_names, waiting_times, color='skyblue')
    plt.xlabel('Jobs')
    plt.ylabel('Waiting Time')
```

```
    plt.title('Waiting Time Comparison of Jobs')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# Setting up the main window
root = tk.Tk()
root.title("Job Scheduling Algorithm (FCFS)")
root.geometry("400x450")
root.configure(bg="#f0f8ff")

# Title label
title_label = tk.Label(root, text="Job Scheduling using FCFS", font=("Helvetica", 16), bg="#f0f8ff")
title_label.pack(pady=10)

# Instructions label
instructions_label = tk.Label(root, text="Enter jobs in format: name arrival_time burst_time (one per line)", bg="#f0f8ff")
instructions_label.pack(pady=5)

# Input for jobs
job_entry = tk.Text(root, height=10, width=40)
job_entry.pack(pady=5)

# Button to calculate scheduling
calculate_button = tk.Button(root, text="Calculate Scheduling", command=calculate_scheduling, bg="#4CAF50", fg="white")
calculate_button.pack(pady=15)
```

```
# Label to display the result
result_label = tk.Label(root, text="", font=("Helvetica", 12), bg="#f0f8ff", justify=tk.LEFT)
result_label.pack(pady=5)


# Run the application
root.mainloop()
```

Code Explanation:-

1. **Job Class**: Represents each job with its name, arrival time, and burst time.
2. **Merge Sort Function**: Recursively sorts the list of jobs based on arrival time.
3. **Calculate Scheduling Function**: Parses user input, sorts jobs, calculates waiting and turnaround times, and displays results.
4. **Plotting Function**: Uses Matplotlib to visualize waiting times as a bar chart.
5. **GUI Setup**: Constructs the main window and elements for user interaction.

Output:-

The application generates the following outputs:

1. **Text Output**: Displays waiting and turnaround times for each job in a readable format.
2. **Graphical Output**: A bar chart visualizing the waiting times for each job.

Example Output

If the user inputs:

A 0 5

B 1 3

C 2 8

The application might output:

Job A: Waiting Time = 0, Turnaround Time = 5

Job B: Waiting Time = 5, Turnaround Time = 8

Job C: Waiting Time = 8, Turnaround Time = 16

And a bar chart visualizing these waiting times will be displayed.

CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

Job Scheduling using FCFS

Enter jobs in format: name arrival_time burst_time (one per line)

```
A 0 5
B 1 3
c 2 8
```

Calculate Scheduling

Job A: Waiting Time = 0, Turnaround Time = 5
Job B: Waiting Time = 5, Turnaround Time = 8
Job c: Waiting Time = 8, Turnaround Time = 16

Conclusion:-

The job scheduling application successfully demonstrates the FCFS algorithm with efficient job sorting using Merge Sort. The user-friendly interface and visual representation of results enhance usability and understanding of job scheduling metrics. This project not only serves practical purposes in job management but also provides a foundation for further exploration of more complex scheduling algorithms.

Future Framework:-

To enhance this application further, the following features could be implemented:

- **Additional Scheduling Algorithms**: Incorporate other algorithms like Round Robin, Shortest Job First, or Priority Scheduling to compare performance.
- **Save/Load Functionality**: Allow users to save job configurations and load them later.
- **Improved Visualization**: Offer more detailed graphs and statistics regarding job performance metrics.

- **Error Handling Enhancements**: Implement more robust input validation and error messages to guide users.

Learning Outcomes:-

- Through this project, the following learning outcomes have been achieved:
- **Understanding Job Scheduling**: Gained insight into how job scheduling algorithms function and their importance in computing.
- **Sorting Algorithms**: Learned to implement the Merge Sort algorithm and understand its application in sorting data.
- **GUI Development**: Acquired skills in creating graphical user interfaces using Tkinter, enhancing user interaction.
- **Data Visualization**: Developed the ability to visualize data using Matplotlib, making it easier to present information effectively.
- **Programming Skills**: Enhanced overall programming skills in Python, particularly in handling user input, data processing, and algorithm implementation.