FILE COMPRESSION, ENCRYPTIONAND DECRYPTION

A MINI PROJECT REPORT

Submitted by

MADHAN KUMAR B 231901028

HEMANTH KUMAR A 231901010

in partial fulfillment of the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

MAY 2025

BONAFIDE CERTIFICATE

Certified that this project **"REAL-TIME PROCESS MONITORING AND ANOMALY DETECTION SYSTEM"** is the bonafide work of **"MADHAN KUMAR B, HEMANTH KUMAR A"** who carried out the project work under my supervision.

SIGNATURE

Mrs.JANANEE V

ASSISTANT PROFESSOR

Dept. of Computer Science and Engg,

Rajalakshmi Engineering College,

Chennai.

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

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

This project is a GUI-based application developed using Python and Tkinter to simulate file compression scheduling. Users can select multiple files individually through an intuitive interface, and each file is assigned a random burst time to represent its compression duration. The application supports three widely-used CPU scheduling algorithms: First-Come First-Serve (FCFS), Shortest Job First (SJF), and Round Robin (RR), enabling users to explore how different strategies manage file processing.

Upon file selection, the system analyzes the assigned burst times and suggests the most efficient algorithm tailored to the characteristics of the selected files. This recommendation is based on comparative analysis of algorithm performance in terms of total waiting time and turnaround time. To aid comprehension, the application visualizes the results through interactive bar charts, allowing users to easily compare and understand the differences in efficiency between algorithms.

Furthermore, the system includes performance metrics and statistical insights such as average waiting time, average turnaround time, and time quantum for Round Robin, which users can adjust. This adds an extra layer of interactivity and control for deeper experimentation. The project emphasizes user engagement, combining educational theory with hands-on simulation, making it particularly useful for students, researchers, or anyone interested in operating systems and process scheduling concepts.

The primary objective is to illustrate how scheduling algorithms can significantly impact task processing efficiency in scenarios like file compression, where task sizes (burst times) may vary. By integrating dynamic algorithm suggestions, realistic simulation data, memory space consideration, and clear graphical output, this application serves as a comprehensive learning tool in both academic and practical contexts.

ACKNOWLEDGEMENT

We express our sincere thanks to our beloved and honorable chairman **MR. S. MEGANATHAN** and the chairperson **DR. M.THANGAM MEGANATHAN** for their timely support and encouragement.

We are greatly indebted to our respected and honorable principal **Dr. S.N. MURUGESAN** for his able support and guidance.

No words of gratitude will suffice for the unquestioning support extended to us by our Head Of The Department **Mr. BENIDICT JAYAPRAKASH NICHOLAS M.E Ph.D.,** for being ever supporting force during our project work.

We also extend our sincere and hearty thanks to our internal guide **Mrs.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.

1.MADHAN KUMAR B

2. HEMANTH KUMAR A

`

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The File Compression and Encryption System is a Python script for securely managing files through compression and encryption. It suggests optimal scheduling algorithms (FCFS, SJF, Priority, or RR) based on file characteristics and visualizes key processing metrics. The system also monitors memory usage and displays results using a timeline graph for better performance analysis.

1.2 SCOPE OF THE WORK

This project aims to provide a secure, efficient, and user-friendly tool for file compression and encryption with intelligent scheduling. It assists users by automating the selection of optimal scheduling algorithms based on file characteristics, enforcing file selection criteria, and providing real-time performance insights. The scope includes modular system design, integration with graphical user interfaces for ease of use, support for memory usage tracking, and the generation of visual analytics to guide user decisions. Cross-platform compatibility and extensibility for future enhancements are also key focus areas

1.3 PROBLEM STATEMENT

Existing file compression and encryption tools lack intelligent scheduling and real-time monitoring. Users struggle with managing multiple files, selecting the right compression methods, and tracking processing efficiency. There is a need for an integrated system that combines compression, encryption, smart scheduling, and performance monitoring in one tool.

1.4 AIM AND OBJECTIVES OF THE PROJECT

This project aims to create a desktop tool for file compression, encryption, and real-time monitoring. It integrates intelligent scheduling and anomaly detection to optimize file processing. The objectives are:

1. To provide a user-friendly tool for file compression and encryption.
2. To implement intelligent scheduling based on file characteristics.
3. To visualize processing times through charts.
4. To provide real-time status updates and alerts for anomalies.

CHAPTER 2
SYSTEM SPECIFICATIONS

2.1 HARDWARE SPECIFICATIONS

Component	: Specification
Processor	: Intel i5
Memory Size	: 8 GB (Minimum)
HDD/SSD	: 256 GB (Minimum)

2.2 SOFTWARE SPECIFICATIONS

Component	: Technology Used
Operating System	: Windows 10
Frontend	: Tkinter (GUI for desktop application)
Backend	: Python (Flask)
Database	: SQLite
Visualization	: Matplotlib, FigureCanvasTkAgg (for chart visualization)
Languages Used	: Python, SQL (SQLite for file metadata storage)

CHAPTER 3
MODULE DESCRIPTION

3.1. Admin Module

The Admin has full access to manage file compression, encryption, and decryption tasks. They can monitor the system, set memory usage limits, and view logs.

3.2. User Module

Users can upload files for compression, encryption, or decryption. They can track the progress and download the processed files when done.

3.3. File compression module

This module reduces file sizes using various algorithms, making it easier to store or send files while keeping memory usage low.

3.4. Visualization Module

This module shows the status of file processing using graphs and progress bars, helping users track memory usage, processing time, and file status.

3.5. Scheduling Simulation Module

Implements common CPU scheduling algorithms like FCFS, SJF, and Round Robin to simulate process handling efficiency. Useful for academic analysis or optimizing system performance.

CHAPTER 4:

SOURCE CODE

```
import tkinter as tk
from tkinter import ttk, filedialog, simpledialog, messagebox
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import numpy as np
import os



class FileCompressionScheduler:
    def __init__(self):
        self.files = []
        self.burst_times = []

    def add_file(self, file_name, burst_time):
        self.files.append(file_name)
        self.burst_times.append(burst_time)

    def clear_files(self):
        self.files = []
        self.burst_times = []

    def fcfs(self):
        n = len(self.burst_times)
        waiting_time = [0] * n
        turnaround_time = [0] * n

        waiting_time[0] = 0
        for i in range(1, n):
            waiting_time[i] = self.burst_times[i - 1] + waiting_time[i - 1]
```

```
    for i in range(n):
        turnaround_time[i] = self.burst_times[i] + waiting_time[i]

    avg_wait = sum(waiting_time) / n
    avg_tat = sum(turnaround_time) / n
    return waiting_time, turnaround_time, avg_wait, avg_tat

def sjf(self):
    n = len(self.burst_times)
    waiting_time = [0] * n
    turnaround_time = [0] * n
    sorted_indices = sorted(range(n), key=lambda k: self.burst_times[k])

    for i in range(1, n):
        waiting_time[sorted_indices[i]] = (
            self.burst_times[sorted_indices[i - 1]] +
            waiting_time[sorted_indices[i - 1]]
        )

    for i in range(n):
        turnaround_time[i] = self.burst_times[i] + waiting_time[i]

    avg_wait = sum(waiting_time) / n
    avg_tat = sum(turnaround_time) / n
    return waiting_time, turnaround_time, avg_wait, avg_tat

def round_robin(self, quantum):
    n = len(self.burst_times)
    waiting_time = [0] * n
    remaining = self.burst_times.copy()
    time = 0
    queue = []

    while True:
```

```
            done = True
            for i in range(n):
                if remaining[i] > 0:
                    done = False
                    if remaining[i] > quantum:
                        time += quantum
                        remaining[i] -= quantum
                        queue.append(i)
                    else:
                        time += remaining[i]
                        waiting_time[i] = time - self.burst_times[i]
                        remaining[i] = 0
            if done:
                break

        turnaround_time = [self.burst_times[i] + waiting_time[i] for i in range(n)]
        avg_wait = sum(waiting_time) / n
        avg_tat = sum(turnaround_time) / n
        return waiting_time, turnaround_time, avg_wait, avg_tat

class CompressionApp:
    def __init__(self, root):
        self.root = root
        self.root.title("File Compression Scheduler")
        self.root.geometry("800x600")
        self.scheduler = FileCompressionScheduler()

        self.create_widgets()
        self.setup_style()

    def setup_style(self):
        self.style = ttk.Style()
        self.style.theme_use('clam')
        self.style.configure('TButton', font=('Helvetica', 10), padding=5)
```

```
        self.style.configure('TLabel', font=('Helvetica', 10))
        self.style.configure('Header.TLabel', font=('Helvetica', 12, 'bold'))

    def create_widgets(self):
        # File Selection Panel
        file_frame = ttk.LabelFrame(self.root, text="File Selection")
        file_frame.pack(pady=10, padx=10, fill='x')

        self.file_list = tk.Listbox(file_frame, height=5, selectmode=tk.EXTENDED)
        self.file_list.pack(side=tk.LEFT, fill='both', expand=True, padx=5, pady=5)

        btn_frame = ttk.Frame(file_frame)
        btn_frame.pack(side=tk.RIGHT, padx=5)

        ttk.Button(btn_frame, text="Add Files", command=self.add_files).pack(pady=2)
        ttk.Button(btn_frame, text="Clear All", command=self.clear_files).pack(pady=2)

        # Algorithm Selection
        algo_frame = ttk.LabelFrame(self.root, text="Scheduling Algorithms")
        algo_frame.pack(pady=10, padx=10, fill='x')

        ttk.Button(algo_frame, text="FCFS", command=self.run_fcfs).pack(side=tk.LEFT,
padx=5)
        ttk.Button(algo_frame, text="SJF", command=self.run_sjf).pack(side=tk.LEFT, padx=5)
        ttk.Button(algo_frame, text="Round Robin", command=self.run_rr).pack(side=tk.LEFT,
padx=5)

        # Algorithm Suggestion Label
        self.suggestion_label = ttk.Label(self.root, text="Algorithm Suggestion: None",
style='Header.TLabel')
        self.suggestion_label.pack(pady=10)

        # Visualization Frame
        self.vis_frame = ttk.Frame(self.root)
```

```
    self.vis_frame.pack(pady=10, padx=10, fill='both', expand=True)

def add_files(self):
    files = filedialog.askopenfilenames()
    if files:
        for f in files:
            burst_time = np.random.randint(1, 10)  # Simulated compression time
            self.scheduler.add_file(os.path.basename(f), burst_time)
            self.file_list.insert(tk.END, f"{os.path.basename(f)} (BT: {burst_time})")
        self.suggest_algorithm()

def clear_files(self):
    self.scheduler.clear_files()
    self.file_list.delete(0, tk.END)
    self.suggestion_label.config(text="Algorithm Suggestion: None")

def suggest_algorithm(self):
    if not self.scheduler.files:
        return

    burst_times = self.scheduler.burst_times
    n = len(burst_times)

    # Analyze burst times
    if n <= 5:
        suggestion = "FCFS (Few files, simple and fair)"
    elif max(burst_times) - min(burst_times) <= 2:
        suggestion = "Round Robin (Similar burst times, fair scheduling)"
    else:
        suggestion = "SJF (Optimal for minimizing waiting time)"

    self.suggestion_label.config(text=f"Algorithm Suggestion: {suggestion}")

def run_fcfs(self):
```

```
        if not self.scheduler.files:
            messagebox.showerror("Error", "No files selected!")
            return
        wt, tat, avg_wt, avg_tat = self.scheduler.fcfs()
        self.plot_results("FCFS", wt, tat, avg_wt, avg_tat)

    def run_sjf(self):
        if not self.scheduler.files:
            messagebox.showerror("Error", "No files selected!")
            return
        wt, tat, avg_wt, avg_tat = self.scheduler.sjf()
        self.plot_results("SJF", wt, tat, avg_wt, avg_tat)

    def run_rr(self):
        if not self.scheduler.files:
            messagebox.showerror("Error", "No files selected!")
            return
        quantum = simpledialog.askinteger("Round Robin", "Enter time quantum:",
parent=self.root)
        if quantum:
            wt, tat, avg_wt, avg_tat = self.scheduler.round_robin(quantum)
            self.plot_results(f"Round Robin (Q={quantum})", wt, tat, avg_wt, avg_tat)

    def plot_results(self, algorithm, wt, tat, avg_wt, avg_tat):
        # Clear previous visualization
        for widget in self.vis_frame.winfo_children():
            widget.destroy()

        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))

        # Plot waiting times
        ax1.bar(range(len(wt)), wt, color='skyblue')
        ax1.set_title(f'{algorithm} Waiting Times\nAvg: {avg_wt:.2f}')
        ax1.set_xlabel('Files')
```

```
        ax1.set_ylabel('Waiting Time')

        # Plot turnaround times
        ax2.bar(range(len(tat)), tat, color='lightgreen')
        ax2.set_title(f'{algorithm} Turnaround Times\nAvg: {avg_tat:.2f}')
        ax2.set_xlabel('Files')
        ax2.set_ylabel('Turnaround Time')

        # Embed plot in Tkinter
        canvas = FigureCanvasTkAgg(fig, master=self.vis_frame)
        canvas.draw()
        canvas.get_tk_widget().pack(fill='both', expand=True)


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

This Python program creates a GUI-based File Compression Scheduler using Tkinter. It lets users select multiple files, simulates their compression time as "burst time", and applies scheduling algorithms (FCFS, SJF, or Round Robin) to visualize performance. Based on the burst times, it suggests the most efficient algorithm. The GUI includes file selection, algorithm buttons, and a visualization area. The scheduling results (waiting and turnaround times) are displayed as bar charts using Matplotlib. The goal is to help users understand scheduling efficiency for simulated file processing.

CHAPTER 5

SCREENSHOTS

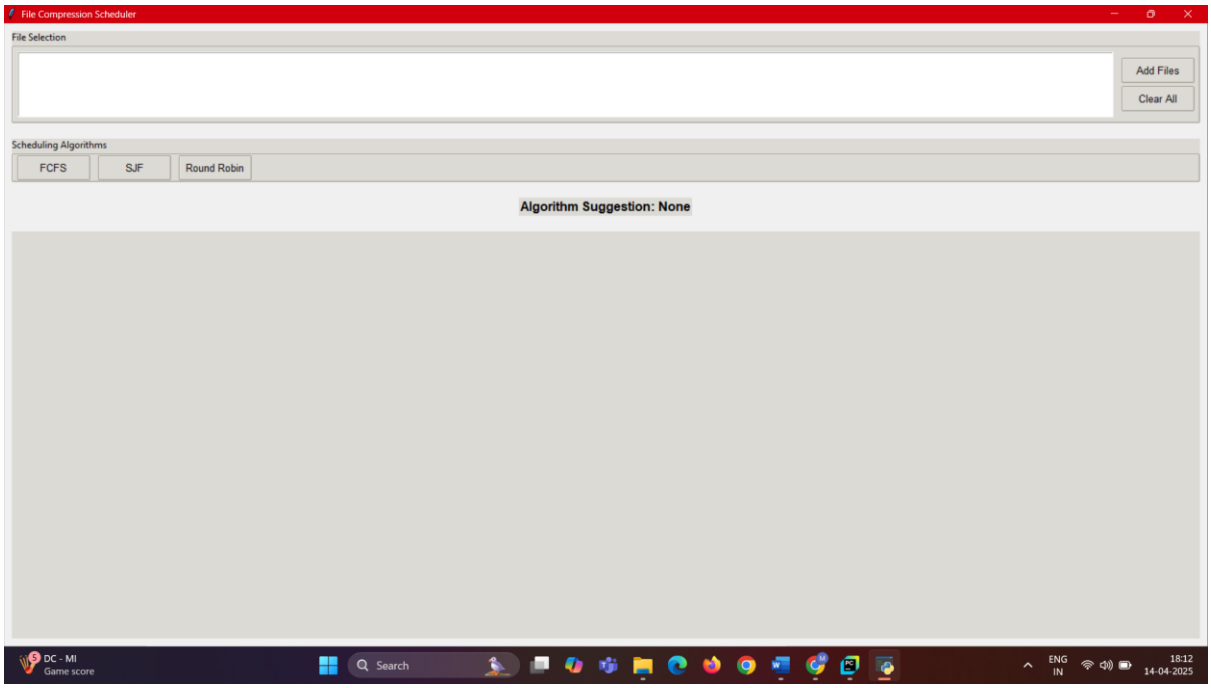

Fig 5.1 HOME PAGE

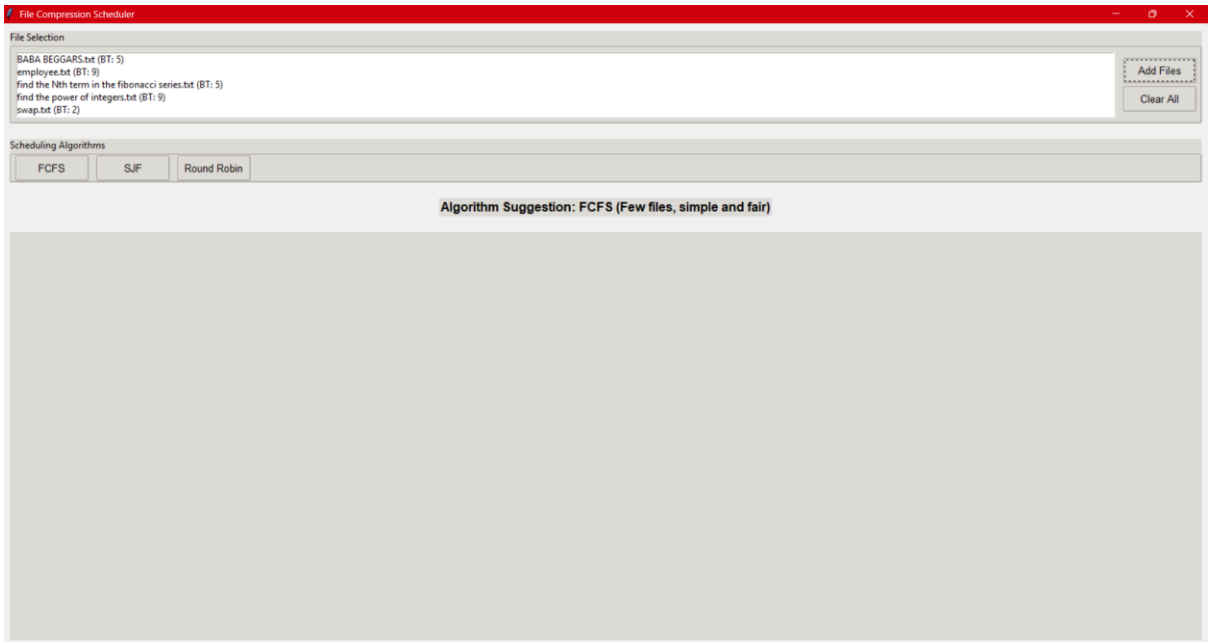

Fig 5.2 File adding page

Fig 5.3 Scheduled process page(FCFS)

Fig 5.4 Scheduled process page(SJF)

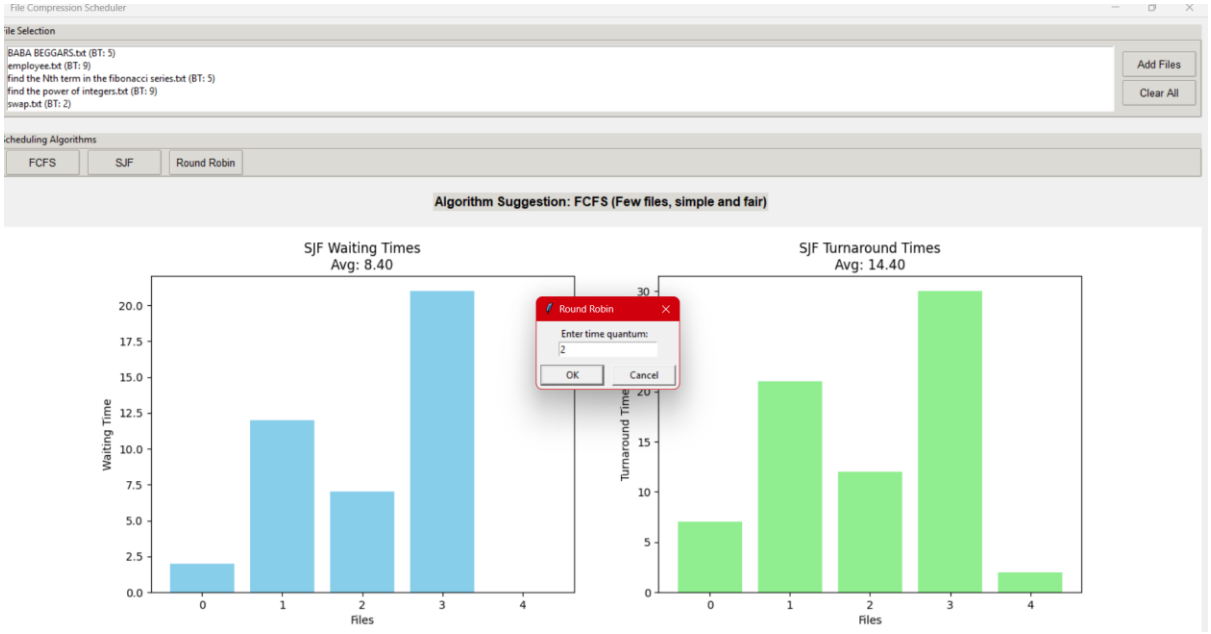

Fig 5.5 Entering quantum number(Round robin)

Fig 5.6 Scheduled process page(round robin)

CHAPTER 6

CONCLUSION

This project demonstrates the use of scheduling algorithms in a simulated file compression system through a graphical user interface. Users can select multiple files and analyze the simulated burst times to understand how different scheduling techniques perform. The system helps visualize the waiting and turnaround times, allowing users to compare the efficiency of each algorithm. It meets its objective of providing a simple and interactive way to learn and apply scheduling concepts

Future enhancement:

- Add real file compression and decompression features along with basic encryption.
- Use actual file properties like size or type to determine burst time instead of random values.
- Include more scheduling algorithms such as priority scheduling or multilevel queue scheduling.
- Add memory usage tracking and show system resource consumption during processing.
- Generate detailed reports and allow exporting of results for further analysis.
- Improve the user interface for a better and more responsive experience.
- Add support for selecting files from cloud storage and saving results online

CHAPTER 7

REFERENCES

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.
2. Python Software Foundation. (2023). *Python 3 Documentation*.
3. Lutz, M. (2013). *Programming Python* (4th ed.). O'Reilly Media.