FILE MANAGER

A MINI PROJECT REPORT

Submitted by

LAKSHMIKANTH K 231901026

NITHEESH K K 231901035

MADHESH M A 231901029

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

APRIL 2025

BONAFIDE CERTIFICATE

Certified that this project **“FILE MANAGER”** is the bonafide work of **“LAKSHMIKANTH K, NITHEESH K K, MADHESH M A”** who carried out the project work under my supervision.

SIGNATURE

Mrs. V JANANEE

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 mini project is a **File Manager with Integrated CPU Scheduling Visualizer**, developed using Python's Tkinter library for GUI. The application allows users to perform essential file management tasks such as listing files, creating, deleting, renaming files, and playing video files within a selected directory. Additionally, it features a process scheduling module that simulates and visualizes two classic CPU scheduling algorithms—FIFO and SJF—using Gantt charts. This combination of file operations and scheduling visualization makes the tool both educational and practical, ideal for understanding basic operating system concepts alongside user-friendly file handling.

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** for being an 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. **LAKSHMIKANTH K**
2. **NITHEESH K K**
3. **MADHESH M A**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

This project integrates two essential computing functionalities: file management and CPU process scheduling. It is implemented in Python with a user-friendly GUI using Tkinter. The goal is to provide both a practical file handling tool and a basic simulation of process scheduling techniques

2. SCOPE OF THE WORK

The system provides capabilities to manage files (create, delete, rename, play video files) and simulate CPU scheduling algorithms like FIFO and SJF with visual representations (Gantt chart). This application is particularly useful for understanding OS-level concepts interactively.

3. PROBLEM STATEMENT

Manual simulation of CPU scheduling and file operations can be time-consuming and prone to error. There's a need for a tool that can visualize scheduling behavior and simplify common file system tasks.

4. AIM AND OBJECTIVES OF THE PROJECT

Aim and objectives:

- To build a GUI-based application for managing files in a chosen directory.
- To simulate FIFO and SJF scheduling algorithms.
- To visualize scheduling outputs through Gantt charts.
- To provide a hands-on learning experience of operating system concepts.

CHAPTER 2

SYSTEM SPECIFICATIONS

1. HARDWARE SPECIFICATIONS

Component	: Specification
Processor	: Dual core or Higher
RAM	: 2 GB (Minimum)
Storage	: 100 MB (Minimum)

2. SOFTWARE SPECIFICATIONS

Operating System	: Windows/Linux/ MacOS
Dependencies	: tkinter, matplotlib
Visualization	: GUI
Languages Used	: Python

CHAPTER 3

MODULE DESCRIPTION

1. Main Window and Event Loop

- List files in selected directory
- Create, delete, and rename files
- Play video files using system's default player

2. Process Monitoring Logic

- Stimulates FIFO and SJF scheduling algorithms
- Displays results using a Gantt chart for visualization

CHAPTER 4

SOURCE CODE:

main.py

```
import os
import sys

import subprocess

import tkinter as tk

from tkinter import filedialog, messagebox

from tkinter import ttk

import matplotlib.pyplot as plt


# Simulated Processes

processes = [

("Process 1", 4),

("Process 2", 3),

("Process 3", 5),

("Process 4", 2)

]


# Track selected directory

current_directory = None


# Scheduling Algorithms

def fifo(processes):

order = []

time = 0

for process in processes:
```

```
time += process[1]

order.append((process[0], time))

return order


def sjf(processes):

processes_sorted = sorted(processes, key=lambda x: x[1])

order = []

time = 0

for process in processes_sorted:

time += process[1]

order.append((process[0], time))

return order


def display_schedule(order):

result = "Process Completion Order (Process Name, Finish Time):\n"

for process, finish_time in order:

result += f"{process}: {finish_time}\n"

messagebox.showinfo("Scheduling Result", result)


def draw_gantt_chart(order):

fig, gnt = plt.subplots()

gnt.set_title("Gantt Chart")

gnt.set_xlabel("Time")

gnt.set_ylabel("Processes")

gnt.set_yticks([10 * i for i in range(1, len(order)+1)])

gnt.set_yticklabels([p[0] for p in order])
gnt.grid(True)


start_time = 0

for i, (process, finish_time) in enumerate(order):

burst_time = finish_time - start_time

gnt.broken_barh([(start_time, burst_time)], (10*(i+1)-2, 4), facecolors=('tab:blue'))

start_time = finish_time


plt.tight_layout()

plt.show()


def run_fifo():

order = fifo(processes)

display_schedule(order)

draw_gantt_chart(order)


def run_sjf():

order = sjf(processes)

display_schedule(order)

draw_gantt_chart(order)


# File Management Functions

def update_file_list():

file_list.delete(0, tk.END)

if current_directory and os.path.isdir(current_directory):

for file in os.listdir(current_directory):
```

```
file_list.insert(tk.END, file)


def list_files():

global current_directory

directory = filedialog.askdirectory()

if directory:

current_directory = directory

update_file_list()


def create_file():

global current_directory

if not current_directory:

messagebox.showerror("Error", "Please select a directory first using 'List Files'.")

return

file_path = filedialog.asksaveasfilename(initialdir=current_directory, defaultextension=".txt",
filetypes=[("Text files", ".txt"), ("All files", ".*")])

if file_path:

try:

with open(file_path, 'w') as file:

file.write("")

messagebox.showinfo("Success", "File created successfully")

update_file_list()

except Exception as e:

messagebox.showerror("Error", str(e))


def delete_file():

global current_directory
```

```
if not current_directory:
messagebox.showerror("Error", "Please select a directory first using 'List Files'.")
return
file_path = filedialog.askopenfilename(initialdir=current_directory)
if file_path and os.path.exists(file_path):
confirm = messagebox.askyesno("Confirm Delete", "Are you sure you want to delete this file?")
if confirm:
try:
os.remove(file_path)
messagebox.showinfo("Success", "File deleted successfully")
update_file_list()
except Exception as e:
messagebox.showerror("Error", str(e))


def rename_file():
global current_directory
if not current_directory:
messagebox.showerror("Error", "Please select a directory first using 'List Files'.")
return
old_path = filedialog.askopenfilename(initialdir=current_directory)
if old_path:
new_path = filedialog.asksaveasfilename(initialdir=current_directory)
if new_path:
try:
os.rename(old_path, new_path)
messagebox.showinfo("Success", "File renamed successfully")
```

```
update_file_list()
except Exception as e:
messagebox.showerror("Error", str(e))

# Video Playback
def play_video():
global current_directory
selected = file_list.curselection()
if not selected:
messagebox.showwarning("No Selection", "Please select a video file from the list.")
return

filename = file_list.get(selected[0])
filepath = os.path.join(current_directory, filename)

if not os.path.isfile(filepath):
messagebox.showerror("Error", "File not found.")
return

if not filename.lower().endswith(('.mp4', '.avi', '.mkv', '.mov')):
messagebox.showerror("Unsupported Format", "Please select a valid video file.")
return

try:
if sys.platform.startswith('darwin'): # macOS
subprocess.call(('open', filepath))
```

```
elif os.name == 'nt': # Windows
os.startfile(filepath)
elif os.name == 'posix': # Linux
subprocess.call(('xdg-open', filepath))
except Exception as e:
messagebox.showerror("Error", f"Could not open video: {str(e)}")

# GUI Setup
root = tk.Tk()
root.title("Process Scheduler & File Manager")
root.geometry("900x600")
root.configure(bg="#f0f0f0")

style = ttk.Style()
style.theme_use('default')
style.configure("TButton", font=("Segoe UI", 10), padding=6)
style.configure("TNotebook", tabposition='n', background="#f0f0f0")
style.configure("TNotebook.Tab", padding=(10, 5))
style.configure("TLabel", background="#f0f0f0", font=("Segoe UI", 11))

notebook = ttk.Notebook(root)
notebook.pack(fill='both', expand=True, padx=10, pady=10)

# --- File Manager Tab ---
file_tab = ttk.Frame(notebook)
notebook.add(file_tab, text="📁 File Manager")
```

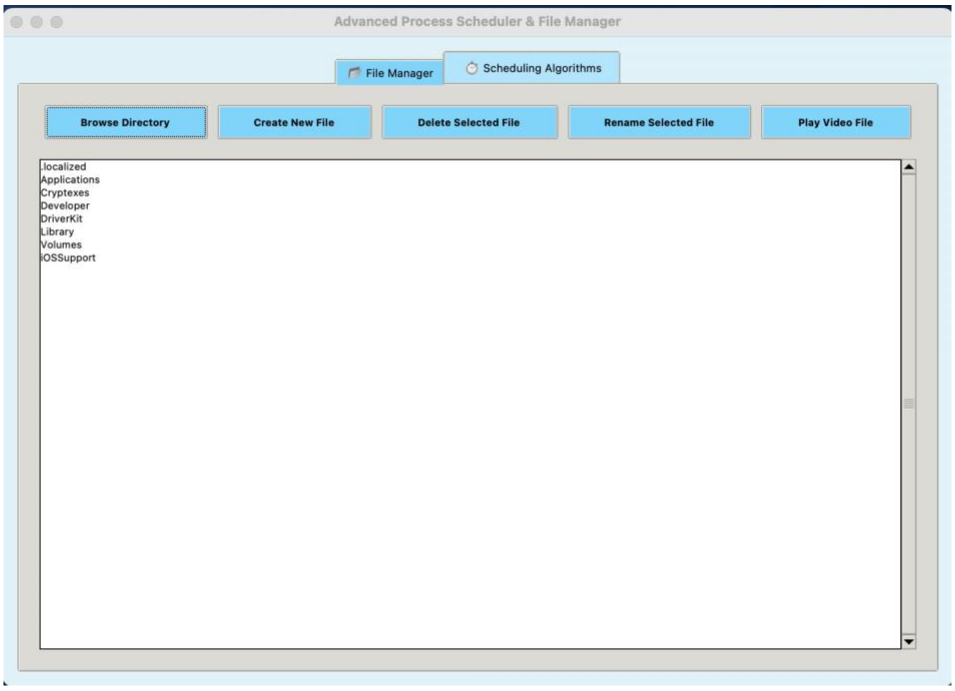
```
file_btn_frame = ttk.Frame(file_tab)

file_btn_frame.pack(pady=15)


file_buttons = [

("List Files", list_files),

("Create File", create_file),

("Delete File", delete_file),

("Rename File", rename_file),

("Play Video", play_video)

]


for i, (text, cmd) in enumerate(file_buttons):

ttk.Button(file_btn_frame, text=text, command=cmd).grid(row=0, column=i, padx=10)


file_list_frame = ttk.Frame(file_tab)

file_list_frame.pack(fill=tk.BOTH, expand=True, padx=20)


scrollbar = ttk.Scrollbar(file_list_frame, orient=tk.VERTICAL)

file_list = tk.Listbox(file_list_frame, height=20, yscrollcommand=scrollbar.set, font=("Segoe UI",
10))

scrollbar.config(command=file_list.yview)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

file_list.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)


# --- Scheduling Tab ---

schedule_tab = ttk.Frame(notebook)
```

```
notebook.add(schedule_tab, text="⚙️ Scheduling Algorithms")


sched_label = ttk.Label(schedule_tab, text="Choose a Scheduling Algorithm:")

sched_label.pack(pady=20)


sched_btn_frame = ttk.Frame(schedule_tab)

sched_btn_frame.pack()


ttk.Button(sched_btn_frame, text="Run FIFO", command=run_fifo).grid(row=0, column=0, padx=20, pady=10)

ttk.Button(sched_btn_frame, text="Run SJF", command=run_sjf).grid(row=0, column=1, padx=20, pady=10)


root.mainloop()
```

CHAPTER 5

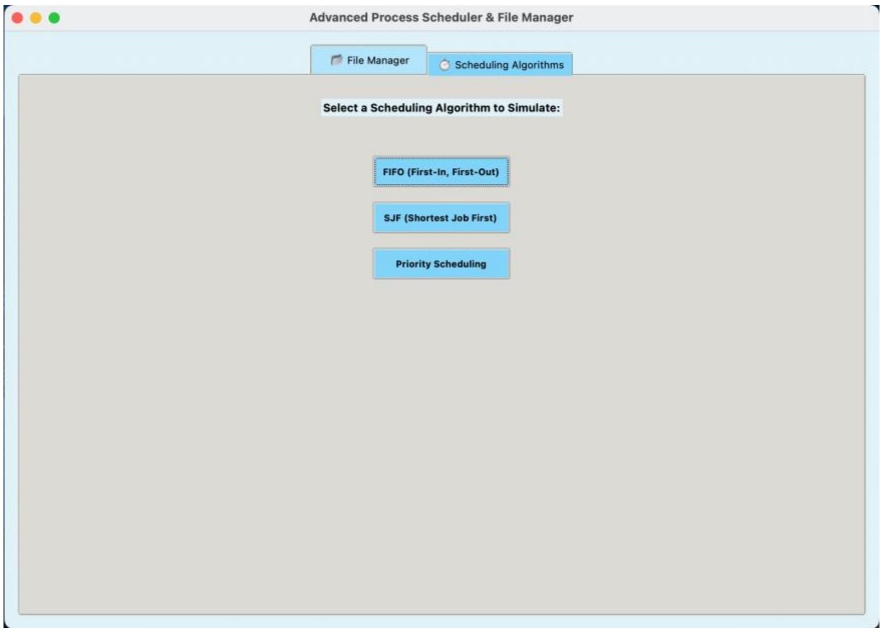
SCREENSHOTS

2.1. ADVANCED PROCESS SCHEDULAR

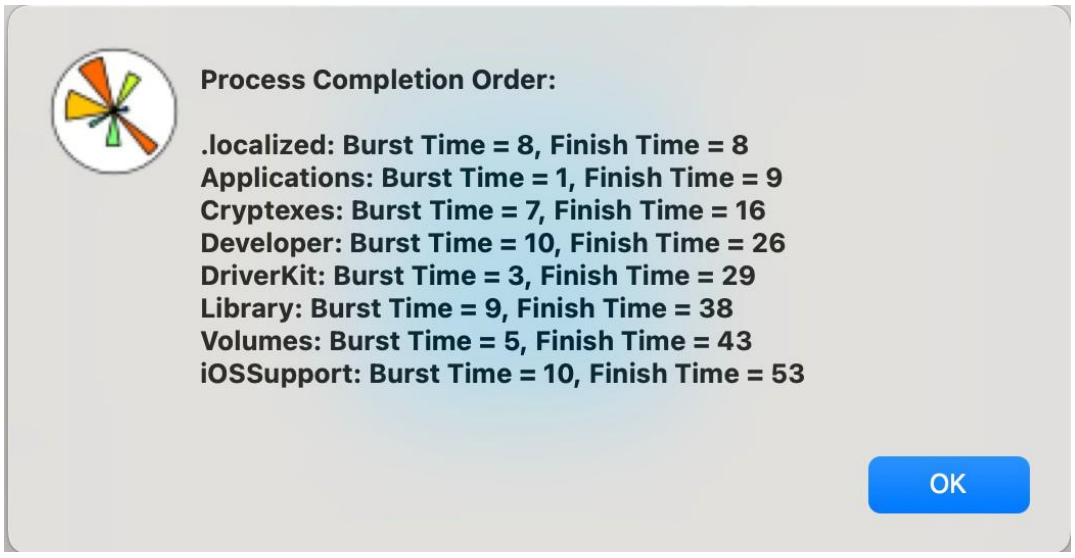


```
.localized
Applications
Cryptexes
Developer
DriverKit
Library
Volumes
iOSSupport
```

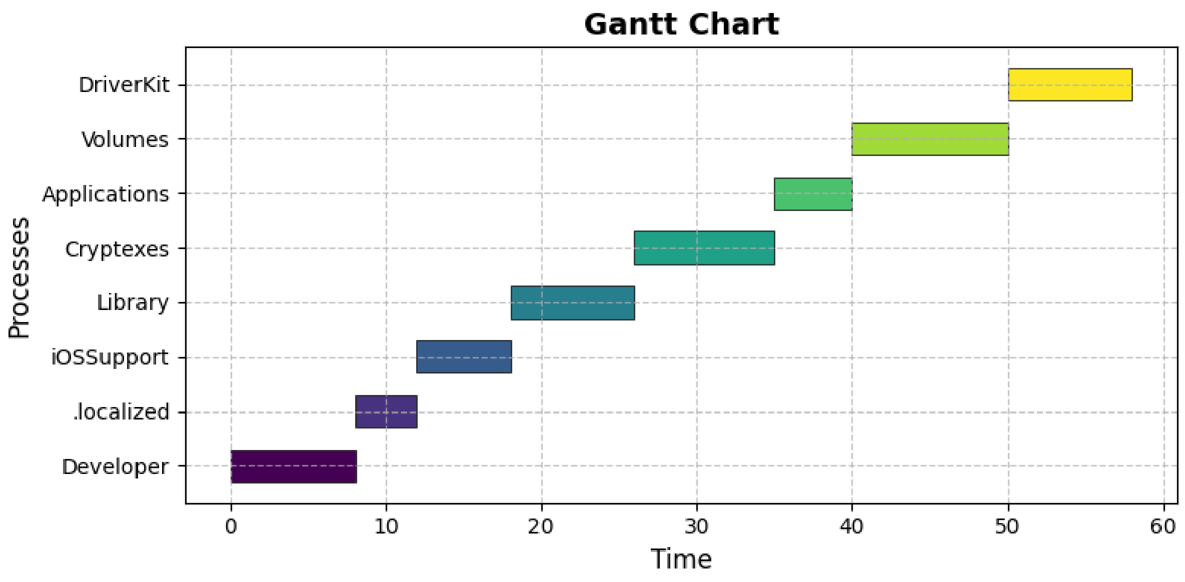
2.2. SCHEDULING ORDER ALGORITHM



Advanced Process Scheduler & File Manager
File Manager
Scheduling Algorithms
Select a Scheduling Algorithm to Simulate:
FIFO (First-In, First-Out)
SJF (Shortest Job First)
Priority Scheduling



```
Process Completion Order:

.localized: Burst Time = 8, Finish Time = 8
Applications: Burst Time = 1, Finish Time = 9
Cryptexes: Burst Time = 7, Finish Time = 16
Developer: Burst Time = 10, Finish Time = 26
DriverKit: Burst Time = 3, Finish Time = 29
Library: Burst Time = 9, Finish Time = 38
Volumes: Burst Time = 5, Finish Time = 43
iOSSupport: Burst Time = 10, Finish Time = 53
```



CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

Conclusion:

The project successfully demonstrates basic file handling and process scheduling concepts using an intuitive GUI. In the future, more scheduling algorithms (e.g., Round Robin, Priority Scheduling) and file operations (copy/move files, directory creation) can be integrated to enhance functionality.

CHAPTER 7

REFERENCES

1. **Mark Lutz**, *Programming Python*, 4th Edition, O'Reilly Media, 2010.
2. **David Beazley and Brian K. Jones**, *Python Cookbook*, 3rd Edition, O'Reilly Media, 2013
3. **Benjamin Root**, *The Book of Matplotlib: Build Data Visualizations with Python*, No Starch Press, 2021.