ACADEMIC TASK-2

CSE326

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Name: Abhijeet Rathore

Registration number: 12317820

Roll No.: 41

Section: K23CW

Submitted to :

Gagandeep Kaur Mam

LOVELY PROFESSIONAL UNIVERSITY

DECLARATION

I, Abhijeet Rathore, a student of Bachelor of Technology under CSE discipline at Lovely Professional University, Punjab, hereby declare that all the information furnished in this project report is based on my own work and is genuine

Your name : Abhijeet Rathore

Registrartion Number: 12317820

Real-Time Process Monitoring Dashboard

1. Project Overview

The Real-Time Process Monitoring Dashboard is designed to provide administrators with a clear, graphical view of system performance and resource utilization. The dashboard will offer real-time data on active processes, CPU usage, and memory consumption, enabling prompt issue detection and better process management. The goal is to improve system efficiency by identifying bottlenecks and ensuring critical resources are available as needed.

2. Module-Wise Breakdown

The project is divided into the following modules:

1. GUI Module: This module builds the user interface using Tkinter, allowing real-time display and interaction with system metrics.
2. Monitoring Module: Uses the psutil library to fetch live data about CPU, memory, and running processes.
3. Visualization Module: Displays data graphically with bar charts or tables for clear and intuitive insight into the system's performance.

3. Functionalities

- Display real-time CPU and memory usage.
- List all currently running processes along with their PID, CPU%, and Memory%.
- Refresh data at regular intervals.
- Option to terminate a selected process.
- Provide alert indicators for high CPU or memory usage.

4. Technology Used

- Programming Languages: Python
- Libraries and Tools: Tkinter, psutil, matplotlib
- Other Tools: GitHub for version control

5. Flow Diagram

The flow diagram describes how the GUI interacts with the backend system metrics and renders them in real time.

6. Revision Tracking on GitHub

• Repository Name: RealTime-Process-Monitor
• GitHub Link: https://github.com/Abhi-ios4940/Process-Management.git

7. Conclusion and Future Scope

The dashboard serves as an effective tool for monitoring and managing system resources in real-time. In the future, this tool can be extended with logging capabilities, remote monitoring, mobile interface, and integration with system alerts or automation tools to take corrective actions automatically.

8. References

- Python Documentation: https://docs.python.org
- psutil Library: https://pypi.org/project/psutil/
- Tkinter Guide: https://docs.python.org/3/library/tkinter.html

Appendix

A. AI-Generated Project Elaboration/Breakdown Report

This report was partially generated using AI to enhance structure, formatting, and explanations.

B. Problem Statement

Create a graphical dashboard that displays real-time information about process states, CPU usage, and memory consumption. The tool should allow administrators to manage processes efficiently and identify potential issues promptly.

C. Solution/Code

```
import tkinter as tk

from tkinter import ttk

import psutil

import matplotlib.pyplot as plt

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

import threading

import time
```

```
class ProcessDashboard:
    def __init__(self, root):
        self.root = root
        self.root.title("Real-Time Process Monitoring Dashboard")
        self.root.geometry("900x600")


        self.create_widgets()
        self.update_dashboard()


    def create_widgets(self):
        # Title
        title = tk.Label(self.root, text="Real-Time Process Monitoring Dashboard",
font=("Helvetica", 16, "bold"))
        title.pack(pady=10)


        # CPU and Memory Usage Frame
        usage_frame = tk.Frame(self.root)
        usage_frame.pack(pady=10)


        self.cpu_label = tk.Label(usage_frame, text="CPU Usage: ", font=("Helvetica", 12))
        self.cpu_label.grid(row=0, column=0, padx=10)


        self.mem_label = tk.Label(usage_frame, text="Memory Usage: ", font=("Helvetica", 12))
        self.mem_label.grid(row=0, column=1, padx=10)


        # Process Table
        self.tree = ttk.Treeview(self.root, columns=("PID", "Name", "Memory"),
show="headings")

        self.tree.heading("PID", text="PID")

        self.tree.heading("Name", text="Process Name")

        self.tree.heading("Memory", text="Memory (%)")

        self.tree.column("PID", width=100)

        self.tree.column("Name", width=300)

        self.tree.column("Memory", width=100)

        self.tree.pack(pady=10)


        # Graph Frame

        graph_frame = tk.Frame(self.root)

        graph_frame.pack(pady=10)


        self.fig, self.ax = plt.subplots(figsize=(6, 2.5))

        self.ax.set_title("CPU Usage Over Time")

        self.ax.set_ylim(0, 100)

        self.cpu_data = []


        self.line, = self.ax.plot([], [], color='blue')

        self.canvas = FigureCanvasTkAgg(self.fig, master=graph_frame)

        self.canvas.get_tk_widget().pack()


    def update_dashboard(self):

        # Update CPU and Memory

        cpu_percent = psutil.cpu_percent()
        mem = psutil.virtual_memory().percent

        self.cpu_label.config(text=f"CPU Usage: {cpu_percent}%")
        self.mem_label.config(text=f"Memory Usage: {mem}%")

        # Update CPU graph
        self.cpu_data.append(cpu_percent)
        if len(self.cpu_data) > 30:
            self.cpu_data.pop(0)

        self.line.set_data(range(len(self.cpu_data)), self.cpu_data)
        self.ax.set_xlim(0, len(self.cpu_data))
        self.ax.figure.canvas.draw()

        # Update process table
        for row in self.tree.get_children():
            self.tree.delete(row)

        for proc in psutil.process_iter(['pid', 'name', 'memory_percent']):
            try:
                self.tree.insert('', 'end', values=(proc.info['pid'], proc.info['name'],
f"{proc.info['memory_percent']:.2f}%"))
            except (psutil.NoSuchProcess, psutil.AccessDenied):
                continue

        self.root.after(1000, self.update_dashboard)
```

```
# Run the app

if __name__ == "__main__":

    root = tk.Tk()

    app = ProcessDashboard(root)

    root.mainloop()
```