**ADVANCED RAM VISUALIZATION**

**A MINI PROJECT REPORT**

***Submitted by***

|  |  |
| --- | --- |
| **JANIT** | **231901013** |
| **MOHAMED ARSATH** | **231901030** |
| **UDAYA SANKAR** | **231901058** |

***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 **“ADVANCED RAM VISUALIZATION”** is the bonafide work of **“JANIT, MOHAMED ARSATH, UDAYA SANKAR”** 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**

The **Advanced RAM Visualizer** is a real-time system monitoring application developed in Python, utilizing the psutil library and Tkinter for GUI rendering. This project provides dynamic insights into memory usage with graphical visualization of running processes. It continuously monitors active processes and their corresponding memory and CPU utilization, displaying the top consumers in both list and graphical formats. The modular design separates functionality into monitoring, graphical rendering, and utility components, allowing efficient updates and maintainability. The visualizer also supports sorted process views, dynamic refresh, and a responsive user interface styled through custom themes. It aims to offer a lightweight yet powerful tool for users, developers, and system administrators to monitor system performance and detect memory-intensive processes effectively. monitoring and proactive threat detection.

**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. **JANIT**
2. **MOHAMED ARSATH**
3. **UDAYA SANKAR**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
| **1** | **INTRODUCTION** | **7** |
| **1.1** | Introduction | **7** |
| **1.2** | Scope of the Work | **7** |
| **1.3** | Problem Statement | **7** |
| **1.4** | Aim and Objectives of the Project | **7** |
| **2** | **SYSTEM SPECIFICATIONS** | **8** |
| **2.1** | Hardware Specifications | **8** |
| **2.2** | Software Specifications | **8** |
| **3** | **MODULE DESCRIPTION** | **9** |
| **4** | **CODING** | **10** |
| **5** | **SCREENSHOTS** | **16** |
| **6** | **CONCLUSION AND FUTURE ENHANCEMENT** | **17** |
| **7** | **REFERENCES** | **18** |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO.** | **TITLE** | **PAGE NO.** |
| **5.1** | **REAL-TIME RAM USAGE** | **16** |
| **5.2** | **EACH PROCESS RAM USAGE** | **16** |

**CHAPTER 1**

**INTRODUCTION**

* 1. **INTRODUCTION**

The Advanced RAM Visualizer is a real-time process and memory monitoring tool developed in Python. Using libraries such as psutil and Tkinter, this tool provides a live, user-friendly dashboard for viewing system memory, CPU usage, and top-consuming processes through both text and graphical formats.

* 1. **SCOPE OF THE WORK**

This tool aims to deliver an efficient, cross-platform desktop utility that enables users to monitor RAM and CPU resource usage at the process level. It is particularly useful for system administrators, developers, and users who need to track memory hogs and ensure system stability.

* 1. **PROBLEM STATEMENT**

Default system monitors often provide raw or cluttered data, lacking real-time graphical representation and detailed process-wise insights. There is a need for a lightweight, responsive tool that simplifies RAM usage tracking without overwhelming the user.

* 1. **AIM AND OBJECTIVES OF THE PROJECT**

# Aim:

To develop a modular, real-time RAM and CPU usage visualizer that displays current memory status and resource-hungry processes dynamically .

# Objectives:

* + - Display real-time process memory and CPU usage
    - Visualize usage data using horizontal bar graphs
    - Provide a sorted view of top processes
    - Update the data dynamically at regular intervals

**CHAPTER 2**

**SYSTEM SPECIFICATIONS**

* 1. **HARDWARE SPECIFICATIONS**

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

* 1. **SOFTWARE SPECIFICATIONS**

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

**CHAPTER 3**

**MODULE DESCRIPTION**

# Main Window and Event Loop

* Initializes the Tkinter window
* Sets up the title, styles, and overall layout
* Calls update functions to refresh the graphs and process data

# Process Monitoring Logic

* Uses psutil to gather process data
* Returns top processes sorted by memory usage
* Fetches CPU % and memory % for each process

# Graphical Display of Processes

* Draws horizontal bar graphs for process memory usage
* Labels include process name, memory %, and CPU %
* Uses Canvas for drawing with dynamic resizing.

# Style Configuration

* Defines color schemes and fonts for UI
* Centralizes all UI styling for easier customization

# Utility Functions

* Includes helper methods like truncate\_string
* Keeps process names clean and UI readable

**CHAPTER 4**

**SOURCE CODE:**

# main.py

import tkinter as tk from tkinter import ttk

from process\_monitor import get\_processes from process\_graph import ProcessGraph from styles import set\_dark\_theme

class TaskManagerApp: def init (self, root):

self.root = root

self.root.title("Python Task Manager") self.root.geometry("1000x600") set\_dark\_theme(self.root)

self.tree = ttk.Treeview(root, columns=("PID", "Name", "RAM (MB)", "RAM

%"), show="headings") self.tree.heading("PID", text="PID") self.tree.heading("Name", text="Name")

self.tree.heading("RAM (MB)", text="RAM (MB)") self.tree.heading("RAM %", text="RAM %") self.tree.column("Name", width=250) self.tree.pack(fill="both", expand=True, padx=10, pady=10) self.tree.bind("<Double-1>", self.show\_graph)

self.graph\_window = None self.update\_process\_list()

def update\_process\_list(self):

self.tree.delete(\*self.tree.get\_children()) for proc in get\_processes():

self.tree.insert("", "end", values=proc) self.root.after(1000, self.update\_process\_list)

def show\_graph(self, event): selected = self.tree.focus() if not selected:

return

pid = int(self.tree.item(selected)["values"][0]) name = self.tree.item(selected)["values"][1] if self.graph\_window:

self.graph\_window.destroy() self.graph\_window = tk.Toplevel(self.root)

self.graph\_window.title(f"RAM Usage - {name} (PID {pid})") ProcessGraph(self.graph\_window, pid, name)

if name == " main ": root = tk.Tk()

app = TaskManagerApp(root) root.mainloop()

# process\_graph.py

import psutil

import matplotlib.pyplot as plt

import matplotlib.animation as animation

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

class ProcessGraph:

def init (self, root, pid, name): self.root = root

self.pid = pid self.name = name self.mem\_data = [] self.time\_data = []

self.max\_points = 60 # keep 60 seconds of data

self.fig, self.ax = plt.subplots(facecolor="#1e1e1e") self.ax.set\_title(f"Memory Usage (MB)", color="white") self.ax.set\_facecolor("#2e2e2e") self.ax.tick\_params(colors='white')

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

self.canvas = FigureCanvasTkAgg(self.fig, master=self.root) self.canvas.get\_tk\_widget().pack(fill="both", expand=True)

self.ani = animation.FuncAnimation(self.fig, self.update\_graph, interval=1000)

def update\_graph(self, i): try:

proc = psutil.Process(self.pid)

mem = proc.memory\_info().rss / 1024\*\*2

except (psutil.NoSuchProcess, psutil.AccessDenied): mem = 0

if len(self.mem\_data) >= self.max\_points: self.mem\_data.pop(0) self.time\_data.pop(0)

self.mem\_data.append(mem) self.time\_data.append(len(self.time\_data)) self.line.set\_data(self.time\_data, self.mem\_data) self.ax.relim()

self.ax.autoscale\_view()

# process\_monitor

import psutil

def get\_processes(): processes = []

for proc in psutil.process\_iter(['pid', 'name', 'memory\_info']): try:

pid = proc.info['pid'] name = proc.info['name']

mem\_info = proc.info['memory\_info'] ram\_mb = mem\_info.rss / 1024 / 1024 ram\_pct = proc.memory\_percent()

cpu\_pct = proc.cpu\_percent(interval=None) # non-blocking

processes.append(( pid,

name, round(ram\_mb, 2),

round(ram\_pct, 3),

round(cpu\_pct, 2)

))

except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess): continue

return processes

# styles.py

from tkinter import ttk def set\_dark\_theme(root):

style = ttk.Style(root) style.theme\_use("default") # General background

root.configure(bg="#1e1e1e")

# Treeview styling style.configure("Treeview",

background="#2e2e2e", foreground="white", fieldbackground="#2e2e2e", rowheight=25, bordercolor="#444444", borderwidth=0)

style.configure("Treeview.Heading", background="#1e1e1e", foreground="white", font=("Segoe UI", 10, "bold"))

style.map("Treeview", background=[("selected", "#3a3a3a")], foreground=[("selected", "white")])

# Scrollbar styling (optional) style.configure("Vertical.TScrollbar",

gripcount=0, background="#444444", darkcolor="#3a3a3a", lightcolor="#3a3a3a", troughcolor="#2e2e2e", bordercolor="#2e2e2e", arrowcolor="white")

style.configure("Horizontal.TScrollbar", gripcount=0, background="#444444", darkcolor="#3a3a3a", lightcolor="#3a3a3a", troughcolor="#2e2e2e",

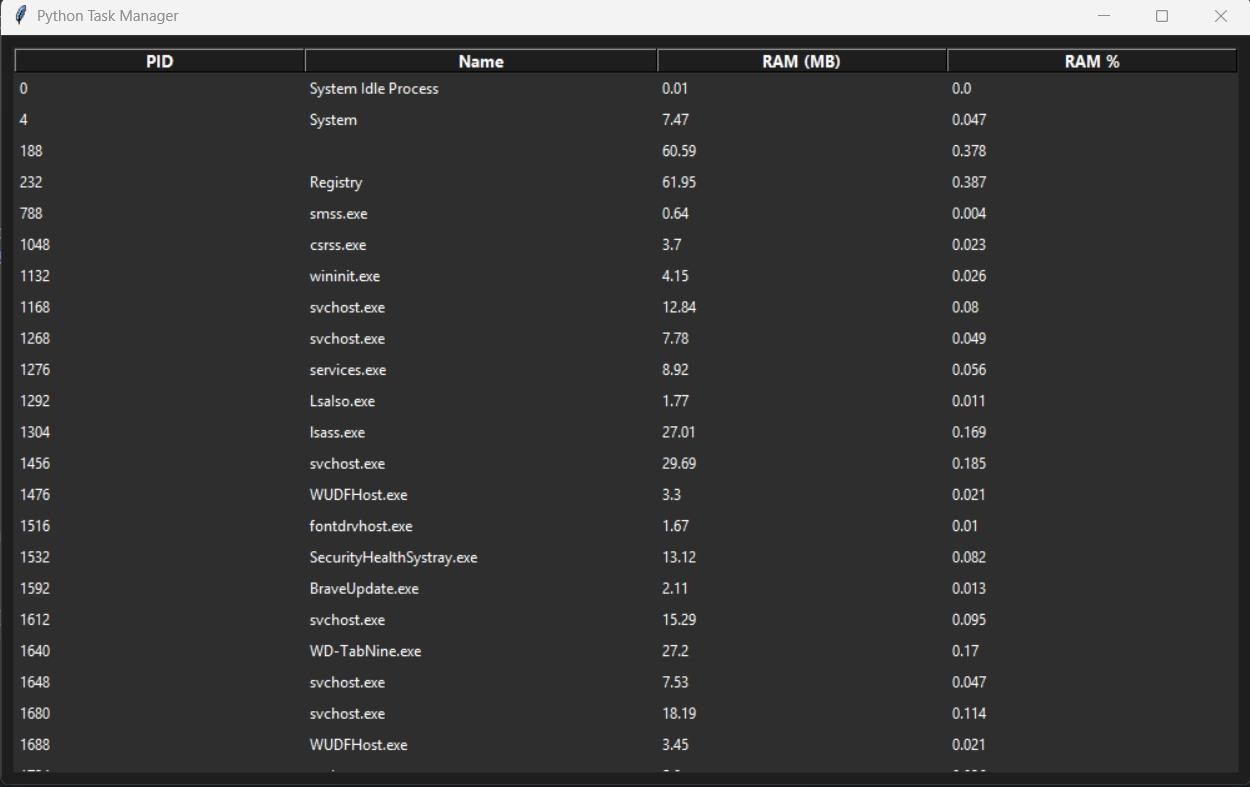
bordercolor="#2e2e2e", arrowcolor="white")

# utils.py

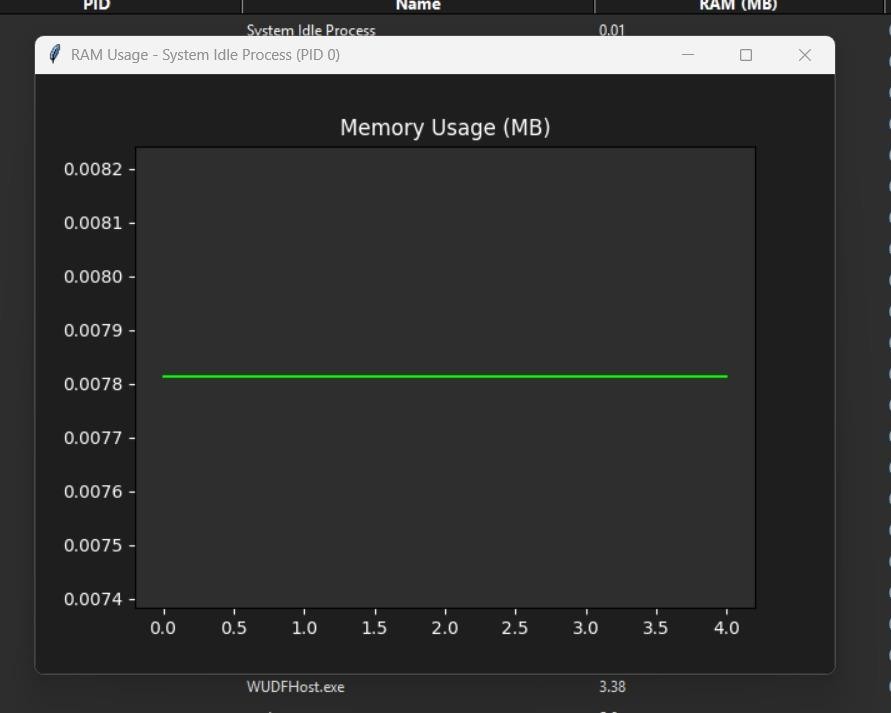
def bytes\_to\_mb(bytes\_val):

return round(bytes\_val / (1024 \* 1024), 2)

**CHAPTER 5 SCREENSHOTS**



* 1. **REAL-TIME RAM USAGE**

****

* 1. **EACH PROCESS RAM USAGE**

**CHAPTER 6 CONCLUSION AND FUTURE ENHANCEMENT**

# Conclusion:

This project successfully demonstrates a real-time RAM and CPU monitoring tool with a graphical interface. It meets the objectives of providing clarity, responsiveness, and modularity.

# Future Enhancements:

* Add CPU graph and pie charts
* Enable system tray minimization
* Export process data to CSV
* Add kill-process functionality
* Integrate historical memory usage tracking

**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.