MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Mini Project Report
of
OPERATING SYSTEMS LAB [CSE 3163]

Resource Utilisation Graph Monitor

SUBMITTED BY

Sanskriti Gupta – 210905150 (23)
Khushi Israni – 210905156 (25)
Aditya Khurana – 210905326 (53)
Vth Sem, Section – A

Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal.
November 2023

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Manipal
20/11/2023

CERTIFICATE

This is to certify that the project titled **Resource Utilisation Graph Monitor** is a record of the bonafide work done by **Sanskriti Gupta (210905150), Khushi Israni (210905156), Aditya Khurana (210905326)** submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in COMPUTER SCIENCE & ENGINEERING of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2022-2023.

Name and Signature of Examiners:

1. **Dr. Sucharitha Shetty, Assistant Professor – Senior Scale, CSE Dept.**

2. **Dr. P B Shanthi, Assistant Professor – Selection Grade, CSE Dept.**

CONTENTS

Abstract

In contemporary computing environments, the efficient utilization of resources is crucial for optimal system performance. This report details the design and implementation of a Resource Utilization Graph Monitor, a comprehensive tool inspired by the functionality of a task manager. This monitor not only visualizes resource usage but also incorporates real-time network information, battery status, disk utilization and task manager. By grounding the development in operating system concepts, this project aims to provide users with a holistic view of their system's health and facilitate a deeper understanding of resource management.

CHAPTER 1:
INTRODUCTION

In the rapidly evolving landscape of computing, the efficient utilization of system resources is paramount for ensuring optimal performance and user satisfaction. A critical aspect of this optimization lies in the ability to monitor and manage resource utilization effectively. Traditional task managers provide insights into CPU and memory usage, but the contemporary computing environment demands a more comprehensive solution. This project introduces a sophisticated Resource Utilization Graph Monitor that transcends conventional boundaries, incorporating real-time data on network activities, battery status, disk utilization and task manager. Rooted in fundamental operating system concepts, this monitor aims to offer users an immersive experience, fostering a deeper understanding of their system's dynamics.

CHAPTER 2:
PROBLEM STATEMENT & OBJECTIVES

The existing paradigm of resource monitoring tools often falls short of providing a holistic view of system health. Conventional task managers primarily focus on CPU and memory usage, leaving gaps in understanding the broader spectrum of resource utilization. Furthermore, the integration of real-time network information, battery status, disk utility and task manager monitoring is often overlooked in contemporary solutions.

The problem at hand is to develop a Resource Utilization Graph Monitor that not only addresses the traditional metrics of CPU, memory, and disk usage but also seamlessly integrates dynamic data on network activities, battery health, disk utility and task manager. This comprehensive

monitoring tool seeks to empower users with a nuanced understanding of their system's behaviour, fostering informed decision-making and a proactive approach to system management.

By marrying sophisticated algorithms with an intuitive user interface, this project aims to bridge the gap between theory and practice, providing users with an insightful and interactive tool to navigate the intricacies of modern computing environments. The solution should be scalable, efficient, and adaptable to diverse operating systems, offering a robust foundation for future enhancements and innovations in resource utilization monitoring.

CHAPTER 3:
ALGORITHM

Initialization:

- Import necessary libraries/modules: **psutil**, **socket**, **tkinter**, **ttk**, **messagebox**, **matplotlib.pyplot**, and **FigureCanvasTkAgg**.
- Define functions:
- **toggle_task_manager**: Toggles the visibility of the task manager window.
- **update_process_list**: Updates the process list displayed in the task manager table.
- **update_utilization_graph**: Updates the utilization graph with real-time data.
- **close_network_info**: Closes the network information window.
- **show_network_info**: Displays network information in a separate window.
- **show_battery_and_bluetooth**: Shows battery and Bluetooth information.
- **show_disk_info**: Shows disk information.
- Create the main application window (**root**) using **tk.Tk()** and set its title to "Task Manager with Utilization Graph."
- Apply styling to the application using **ttk.Style()**.
- Create a frame (**button_frame**) to contain buttons and pack it into the main window.
- Create buttons for:
- Toggling the task manager view.
- Showing network information.
- Showing battery and Bluetooth information.
- Showing disk information.
- Create a Toplevel window (**task_manager**) for the task manager table, set its title, and define its geometry.
- Create a frame (**tree_frame**) for the task manager table.
- Create a Treeview widget (**tree**) to display process information with specified columns.
- Pack the Treeview widget into the frame and pack the frame into the Toplevel window.
- Create a frame (**graph_frame**) for the utilization graph.

- Create a matplotlib figure and canvas for the utilization graph, specifying the size.
- Pack the canvas into the frame and pack the frame into the main window.
- Initialize empty lists (**memory_usage**, **cpu_usage**, **disk_usage**) to store real-time utilization data.
- Start gathering and displaying process information (**update_process_list**).
- Start updating the utilization graph (**update_utilization_graph**).

Functions:

toggle_task_manager Function:

- If the task manager window is currently visible (**winfo_ismapped**), hide it (**withdraw**), else show it (**deiconify**).

update_process_list Function:

- Clear the previous data in the task manager table.
- Get the list of currently running processes.
- Iterate over each process:
- Extract process information (PID, app name, memory utilization, CPU utilization, disk utilization).
- Insert the process information into the task manager table.
- Schedule the function to run again after 2000 milliseconds (2 seconds).

update_utilization_graph Function:

- Gather system resource utilization data (memory, CPU, disk).
- Update the utilization graph with the gathered data.
- If the data exceeds 30 points, remove the oldest data point.
- Schedule the function to run again after 1000 milliseconds (1 second).

show_network_info Function:

- Create a new Toplevel window for network information.
- Create a Text widget for displaying network information.
- Iterate over network interfaces and their addresses, collecting information.
- Insert the collected network information into the Text widget.
- Create a button to close the network information window.

show_battery_and_bluetooth Function:

- Show battery percentage and power status in a message box.

show_disk_info Function:

- Create a new Toplevel window for disk information.
- Iterate over disk partitions, collecting information.
- Insert the collected disk information into the Text widget.
- Create a button to close the disk information window.

close_network_info Function:

- Destroy the network information window.

Main Loop:

- Run the main application loop (**root.mainloop()**).

Code:

```
# Import necessary libraries/modules
import psutil # Library for system monitoring
import socket # Library for network-related functions
import tkinter as tk # GUI library
from tkinter import ttk # Themed Tkinter widgets
from tkinter import messagebox # Message box dialogs in Tkinter
import matplotlib.pyplot as plt # Plotting library
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg # Matplotlib integration with Tkinter


# Function to toggle the visibility of the task manager window
def toggle_task_manager():
if task_manager.winfo_ismapped():
task_manager.withdraw() # Hide the window
else:
task_manager.deiconify() # Show the window

def update_process_list():
# Clear previous data in the tree (GUI component)
tree.delete(*tree.get_children())

# Get a list of currently running processes
process_list = psutil.process_iter()
```

```
# Iterate over each process in the list
for idx, proc in enumerate(process_list):
try:
# Extract process information
pid = proc.pid
app_name = proc.name()
memory_info = f"{proc.memory_info().rss / (1024 * 1024 * 1024):.2f} GB"
# Extract CPU usage information for each core
cpu_info = ", ".join([f"Core {idx + 1}: {util}%" for idx, util in
enumerate(psutil.cpu_percent(percpu=True))])
# Extract disk usage information
disk_info = f"{psutil.disk_usage('/').percent}%"

# Insert the extracted process information into the task manager table (GUI component)
tree.insert("", "end", values=(pid, app_name, memory_info, cpu_info, disk_info))
# Handle exceptions for specific process-related errors
except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
pass

# Schedule the function to run again after 2000 milliseconds (2 seconds)
root.after(2000, update_process_list)


# Function to update the utilization graph
def update_utilization_graph():
# Gather system resource utilization data
memory_usage.append(psutil.virtual_memory().percent)
cpu_usage.append(sum(psutil.cpu_percent(percpu=True)) / psutil.cpu_count())
disk_usage.append(psutil.disk_usage('/').percent)

if len(memory_usage) > 30: # Display the last 30 data points
```

```
memory_usage.pop(0)
cpu_usage.pop(0)
disk_usage.pop(0)

# Update the utilization graph
ax.clear()
ax.plot(memory_usage, label='Memory')
ax.plot(cpu_usage, label='Average CPU')
ax.plot(disk_usage, label='Disk')
ax.legend()
ax.set_xlabel('Time')
ax.set_ylabel('Utilization (%)')
canvas.draw()
root.after(1000, update_utilization_graph) # Update every 1 second


# Function to close the network information window
def close_network_info():
network_info_window.destroy()

# Function to show network information in a separate window
def show_network_info():
try:
global network_info_window # Declare a global variable to track the network information window
network_info_window = tk.Toplevel(root) # Create a new Toplevel window (sub-window) for network information
network_info_window.title("Network Information") # Set the title of the network information window
network_info_text = tk.Text(network_info_window, height=20, width=60) # Create a Text widget for displaying network information
network_info_text.pack(padx=20, pady=20) # Pack the Text widget into the window with padding
```

```
network_info = "" # Initialize an empty string to store network information
interfaces = psutil.net_if_addrs() # Get network interface addresses using psutil
for interface_name, interface_addresses in interfaces.items():
network_info += f"Interface: {interface_name}\n"
for addr in interface_addresses:
if addr.family == socket.AF_INET:
network_info += f" IP Address: {addr.address}\n"
network_info += f" Netmask: {addr.netmask}\n"
elif addr.family == socket.AF_INET6:
network_info += f" IPv6 Address: {addr.address}\n"
network_info += f" Netmask: {addr.netmask}\n"
network_info += "\n"


network_speed = psutil.net_if_stats() # Get network interface statistics
network_info += "Network Speed:\n"
for interface_name, stats in network_speed.items():
network_info += f" Interface: {interface_name}\n"
network_info += f" Speed: {stats.speed} Mbps\n"
network_info += "\n"
network_info_text.insert(tk.END, network_info) # Insert the collected network information into the Text widget
close_button = ttk.Button(
network_info_window, text="Close", command=close_network_info) # Create a button to close the network information window
close_button.pack(pady=10) # Pack the close button into the network information window
except Exception as e:
# Handle exceptions by displaying an error message box with the details of the error
messagebox.showerror("Error", f"An error occurred: {str(e)}")

# Function to show battery and bluetooth information
```

```
def show_battery_and_bluetooth():
try:
battery_info = f"Battery Percentage: {
psutil.sensors_battery().percent}%\n"
battery_info += f"Power Plugged: {
psutil.sensors_battery().power_plugged}\n\n"
messagebox.showinfo("Battery & Bluetooth Info", f"{
battery_info}\n")
except Exception as e:
messagebox.showerror("Error", f"An error occurred: {str(e)}")



def show_disk_info():
try:
disk_info_window = tk.Toplevel(root)
disk_info_window.title("Disk Information")

disk_info_text = tk.Text(disk_info_window, height=20, width=60)
disk_info_text.pack(padx=20, pady=20)

disk_info = ""
partitions = psutil.disk_partitions()
for partition in partitions:
disk_usage = psutil.disk_usage(partition.mountpoint)
disk_info += f"Drive: {partition.device}\n"
disk_info += f"Mountpoint: {partition.mountpoint}\n"
disk_info += f"File System Type: {partition.fstype}\n"
disk_info += f"Total Size: {disk_usage.total /
(1024 * 1024 * 1024):.2f} GB\n"
disk_info += f"Used: {disk_usage.used /
```

```
(1024 * 1024 * 1024):.2f} GB\n"
disk_info += f"Free: {disk_usage.free /
(1024 * 1024 * 1024):.2f} GB\n"
disk_info += f"Percentage Used: {disk_usage.percent}%\n\n"


disk_info_text.insert(tk.END, disk_info)


close_button = ttk.Button(
disk_info_window, text="Close", command=disk_info_window.destroy)
close_button.pack(pady=10)
except Exception as e:
messagebox.showerror("Error", f"An error occurred: {str(e)}")

# Main window
root = tk.Tk()
root.title("Task Manager with Utilization Graph")

# Styling
style = ttk.Style()
style.theme_use('clam')

# Frame to contain buttons
button_frame = ttk.Frame(root)
button_frame.pack()

# Button to toggle task manager view
task_manager_button = ttk.Button(
button_frame, text="Toggle Task Manager", command=toggle_task_manager)
task_manager_button.pack(side=tk.LEFT, padx=5, pady=10)
```

```
# Button to show network information
network_button = ttk.Button(
button_frame, text="Show Network Info", command=show_network_info)
network_button.pack(side=tk.LEFT, padx=5, pady=10)

# Button to show battery and Bluetooth information
battery_bluetooth_button = ttk.Button(
button_frame, text="Show Battery", command=show_battery_and_bluetooth)
battery_bluetooth_button.pack(side=tk.LEFT, padx=5, pady=10)

# Button to show disk information

disk_button = ttk.Button(
button_frame, text="Show Disk Info", command=show_disk_info)
disk_button.pack(side=tk.LEFT, padx=5, pady=10)

# Creating a Toplevel window for the task manager table
task_manager = tk.Toplevel(root)
task_manager.title("Task Manager")
task_manager.geometry("700x400")

# Creating a frame for the task manager table
tree_frame = ttk.Frame(task_manager)

# Creating a treeview to display the process information
tree = ttk.Treeview(tree_frame, columns=(
"PID", "App Name", "Memory Utilization", "CPU Utilization", "Disk Utilization"))
tree.heading("#0", text="Process ID")
tree.heading("#1", text="App Name")
```

```
tree.heading("#2", text="Memory Utilization")
tree.heading("#3", text="CPU Utilization")
tree.heading("#4", text="Disk Utilization")
tree.pack(expand=True, fill=tk.BOTH)
tree_frame.pack(expand=True, fill=tk.BOTH)

# Creating a frame for the utilization graph
graph_frame = ttk.Frame(root)
graph_frame.pack(padx=20, pady=10)
fig, ax = plt.subplots(figsize=(5, 3))
canvas = FigureCanvasTkAgg(fig, master=graph_frame)
canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

memory_usage, cpu_usage, disk_usage = [], [], []

# Start gathering and displaying process information
update_process_list()
update_utilization_graph()

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

CHAPTER 4:
METHODOLOGY

1. Understanding the Project Scope:

- Clarify the objectives and scope of the project, which involves creating a resource utilization monitoring tool with a graphical user interface (GUI).
- Identify key features, including real-time process information and a utilization graph.

2. Selection of Libraries and Tools:

- Choose appropriate libraries and tools for system monitoring, GUI development, and data visualization.
- Select psutil for system monitoring, tkinter for GUI development, and matplotlib for real-time graph plotting.

3. Project Structure Design:

- Design the overall structure of the project, including the main window, buttons, task manager window, and utilization graph.
- Define the interactions between different components and windows.

4. GUI Design and Implementation:

- Develop the main GUI window using tkinter with a set of buttons to perform specific tasks, such as toggling the task manager or displaying network information.
- Create a separate task manager window (Toplevel) containing a treeview widget to display process information.
- Design and implement the network information window, battery, and Bluetooth information window, as well as the disk information window.

5. Task Manager Functionality:

- Implement the function toggle_task_manager() to toggle the visibility of the task manager window.
- Develop update_process_list() to gather and display real-time information about running processes using the psutil library.

6. Utilization Graph Functionality:

- Create update_utilization_graph() to gather and display system resource utilization data, including memory, CPU, and disk.
- Utilize matplotlib for real-time graph plotting, updating every second.

7. Network Information Functionality:

- Implement show_network_info() to create a separate window displaying network information using the psutil and socket libraries.

8. Battery and Bluetooth Information Functionality:

- Develop show_battery_and_bluetooth() to show a message box with battery and Bluetooth information using the psutil library.

9. Disk Information Functionality:

- Create show_disk_info() to display disk information in a separate window, including details about each partition using the psutil library.

10. Styling and Theming:

- Apply styling to the GUI using the ttk module, providing a consistent and visually appealing interface.
- Choose a theme ('clam' in this case) to enhance the overall aesthetics.

11. Testing and Debugging:

- Conduct rigorous testing to ensure the correct functioning of each feature and the overall application.
- Address and debug any identified issues, including handling exceptions gracefully.

12. Documentation:

- Create comprehensive documentation, including comments within the code, to facilitate understanding for future developers or collaborators.
- Document the purpose and functionality of each function.

13. User Interface Optimization:

- Optimize the user interface for a seamless user experience.
- Ensure that buttons and information displays are clear and user-friendly.

14. Security Measures:

- Implement security measures to protect against vulnerabilities and data breaches.
- Ensure that sensitive information is handled securely.

CHAPTER 5:
IMPLEMENTATION

1. Resource Utilization Visualization:

Graphical Representation:
Develop a responsive user interface using a graphical library (e.g., PyQt or Tkinter in Python) to illustrate real-time resource usage through dynamic graphs and charts.

Resource Monitoring Algorithms:
Implement algorithms to monitor CPU usage, memory consumption, and disk utilization, ensuring accuracy and efficiency.

2. Network Information Integration:

Network Monitoring Modules:

Design modules to collect and display network-related data, such as current bandwidth usage, active connections, and network latency.

User Interface Enhancements:

Integrate network information into the user interface, providing users with a comprehensive view of their system's network activities.

3. Battery Status Monitoring:

Battery Information Retrieval:

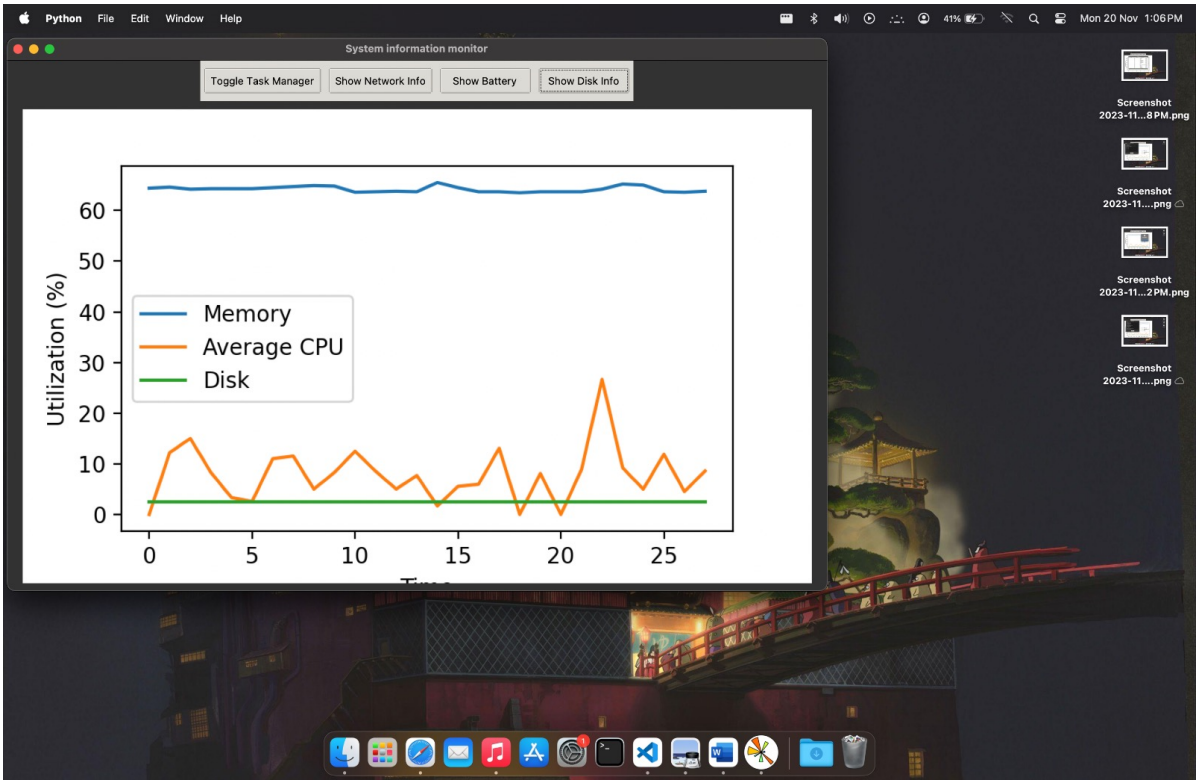
Implement mechanisms to retrieve battery-related data, including current charge, discharge rate, and remaining battery life.

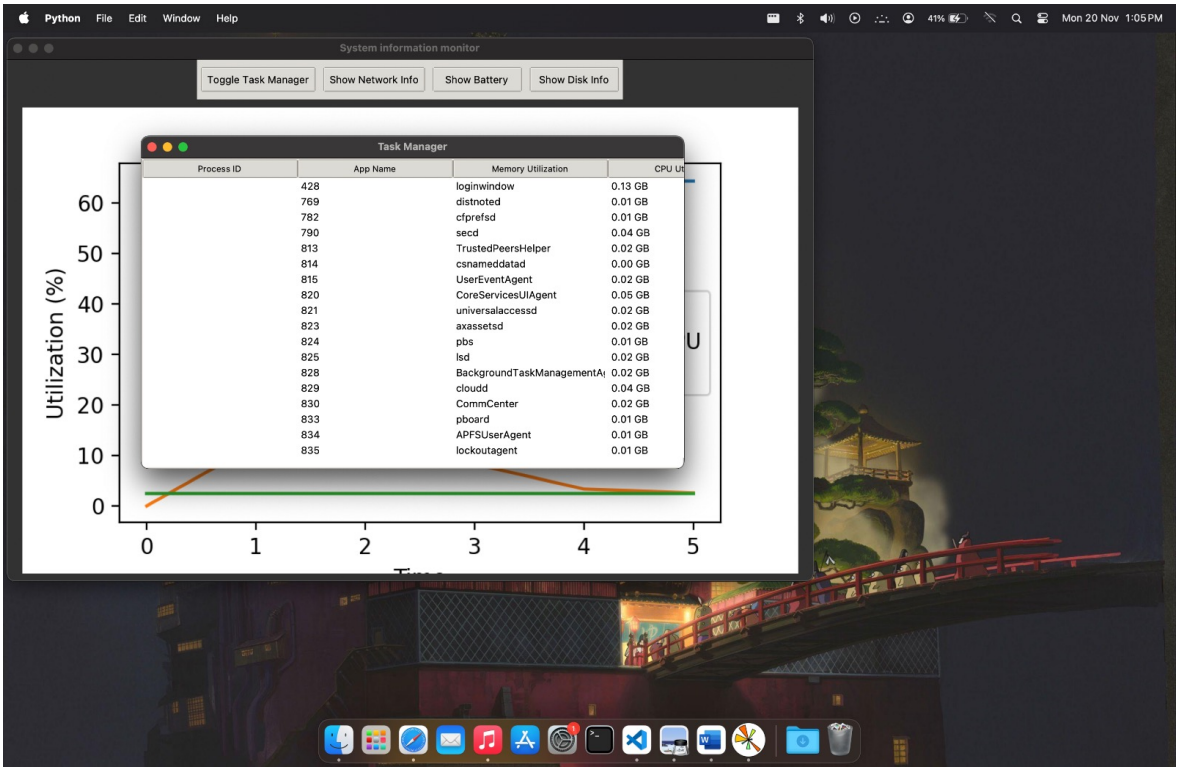
4. User-Friendly Interface:

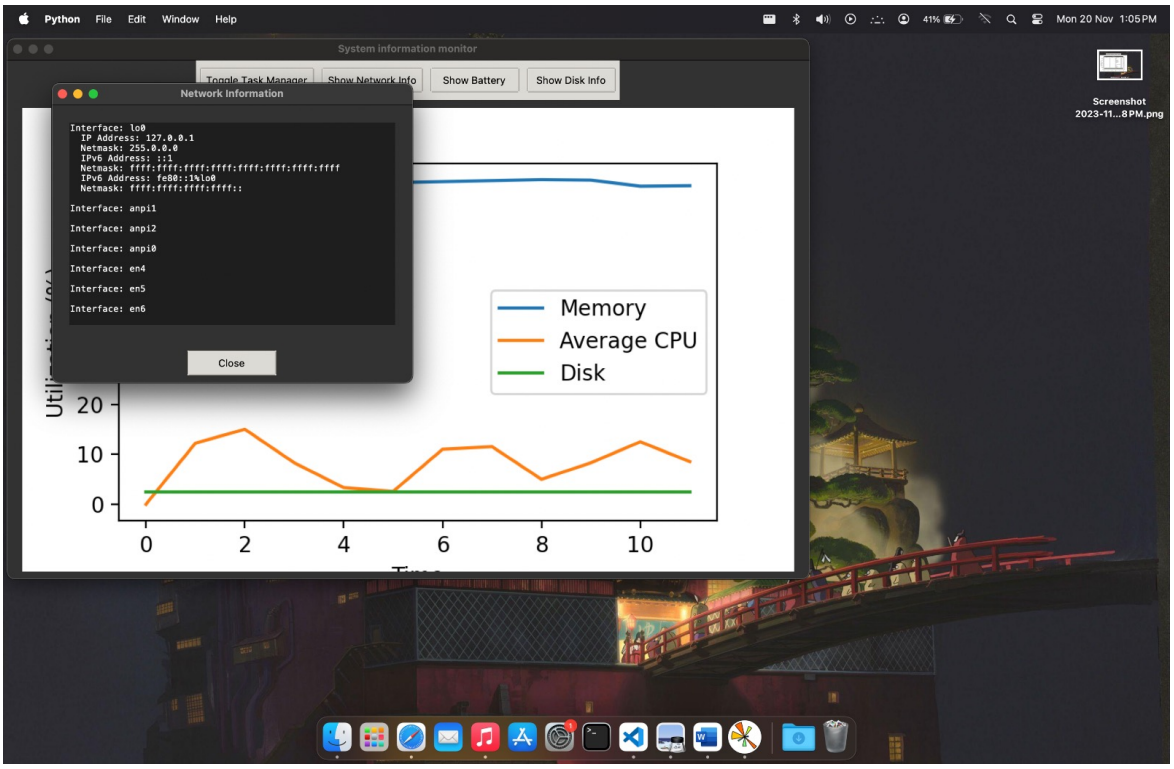
Customizable Dashboards:

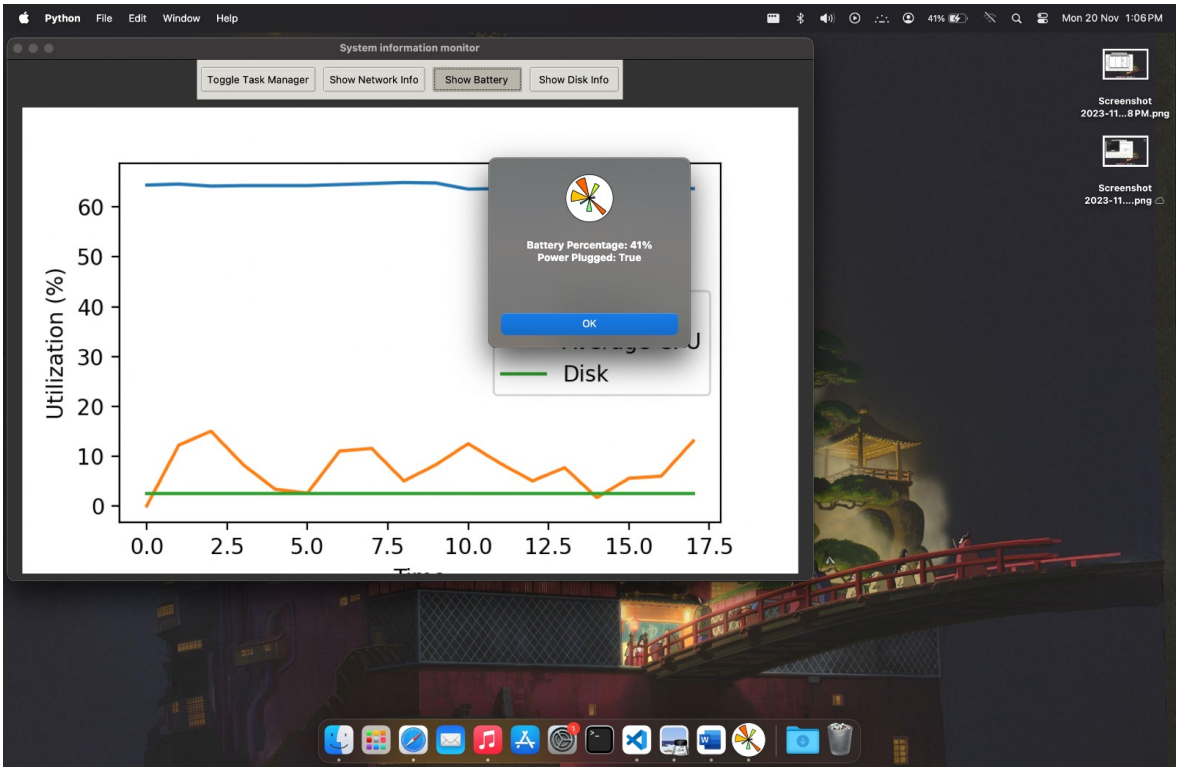
Implement customizable dashboards, allowing users to arrange and prioritize the displayed information based on their preferences.

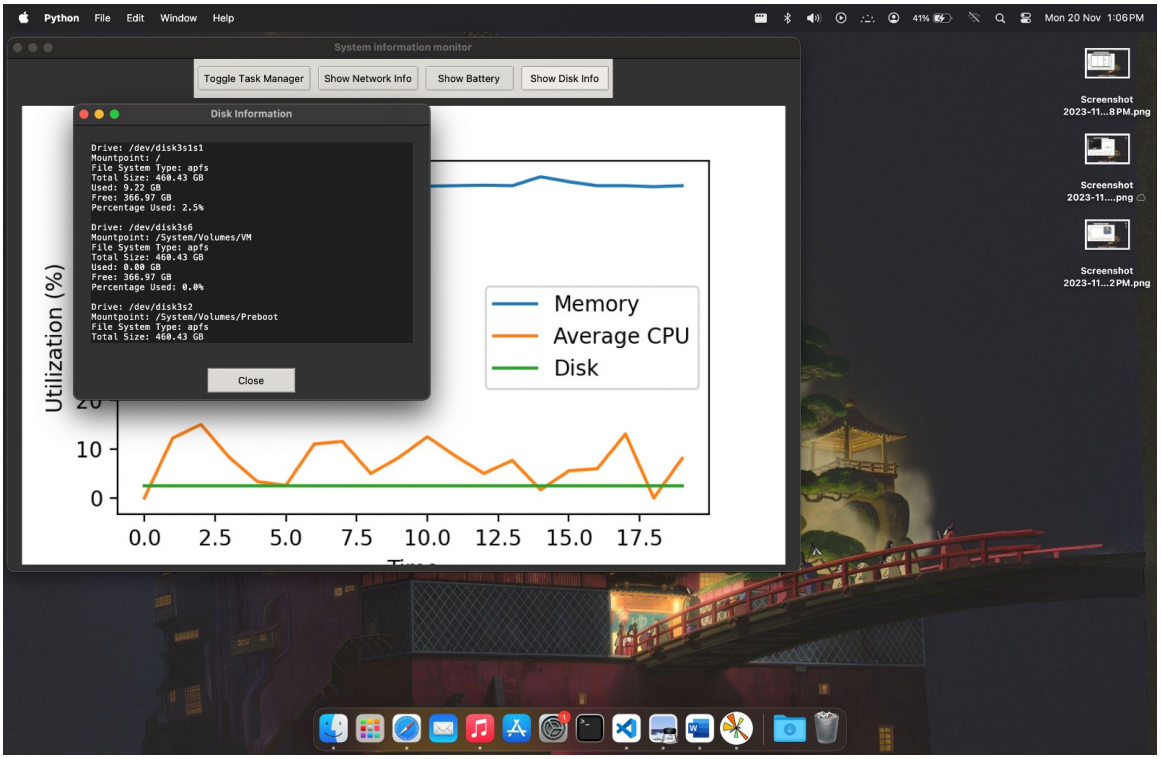
CHAPTER 6:
RESULTS AND SNAPSHOTS




Python File Edit Window Help
Mon 20 Nov 1:05 PM
System information monitor
Toggle Task Manager
Show Network Info
Show Battery
Show Disk Info
Task Manager
Process ID
App Name
Memory Utilization
CPU Ut
428 loginwindow 0.13 GB
769 distnoted 0.01 GB
782 cfprefsd 0.01 GB
790 secd 0.04 GB
813 TrustedPeersHelper 0.02 GB
814 csnameddatad 0.00 GB
815 UserEventAgent 0.02 GB
820 CoreServicesUIAgent 0.05 GB
821 universalaccessd 0.02 GB
823 axassetsd 0.02 GB
824 pbs 0.01 GB
825 lsd 0.02 GB
828 BackgroundTaskManagementA 0.02 GB
829 cloudd 0.04 GB
830 CommCenter 0.02 GB
833 pboard 0.01 GB
834 APFSUserAgent 0.01 GB
835 lockoutagent 0.01 GB
Utilization (%)
Time


Python File Edit Window Help
Mon 20 Nov 1:05 PM
System information monitor
Show Battery
Show Disk Info
Network Information
Interface: lo0
IP Address: 127.0.0.1
Netmask: 255.0.0.0
IPv6 Address: ::1
Netmask: ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff
IPv6 Address: fe80::1%lo0
Netmask: ffff:ffff:ffff:ffff::
Interface: anpi1
Interface: anpi2
Interface: anpi0
Interface: en4
Interface: en5
Interface: en6
Close
Memory
Average CPU
Disk
Screenshot
2023-11...8 PM.png
Time


Python File Edit Window Help
Mon 20 Nov 1:06 PM
System information monitor
Toggle Task Manager
Show Network Info
Show Battery
Show Disk Info
Battery Percentage: 41%
Power Plugged: True
OK
Disk
Utilization (%)
Screenshot 2023-11...8 PM.png
Screenshot 2023-11....png


Python File Edit Window Help
Mon 20 Nov 1:06 PM
System information monitor
Toggle Task Manager
Show Network Info
Show Battery
Show Disk Info
Disk Information
Drive: /dev/disk3s1s1
Mountpoint: /
File System Type: apfs
Total Size: 460.43 GB
Used: 9.22 GB
Free: 366.97 GB
Percentage Used: 2.5%
Drive: /dev/disk3s6
Mountpoint: /System/Volumes/VM
File System Type: apfs
Total Size: 460.43 GB
Used: 0.00 GB
Free: 366.97 GB
Percentage Used: 0.0%
Drive: /dev/disk3s2
Mountpoint: /System/Volumes/Preboot
File System Type: apfs
Total Size: 460.43 GB
Close
Memory
Average CPU
Disk
Utilization (%)
Screenshot 2023-11...8 PM.png
Screenshot 2023-11....png
Screenshot 2023-11...2 PM.png

CHAPTER 7:
CONCLUSION

The Resource Utilization Graph Monitor stands as a robust solution, providing users with a comprehensive understanding of their system's dynamics. By integrating real-time data on network activities, battery status, disk utility and task management, this tool offers a holistic view of resource utilization. The user-friendly interface, customizable dashboards, and alert mechanisms enhance the practical utility, facilitating proactive system management.

Key Achievements:
Comprehensive Insights: The monitor offers a holistic view, covering CPU, memory, disk usage, network activities, battery health and task management.

User Interaction: Customizable dashboards and alerts empower users to tailor their monitoring experience and promptly respond to critical events.

Bridging OS Concepts: The project successfully integrates theoretical OS principles with practical application, enhancing educational value.

CHAPTER 8:
FUTURE SCOPE OF THE PROJECT

- **1. Integration of Additional Metrics:**

  GPU Usage: Incorporate monitoring and visualization of GPU utilization to provide a more comprehensive view of system performance, especially in systems with dedicated graphics cards.
  Storage Metrics: Expand the monitoring capabilities to include detailed metrics on storage devices, such as read and write speeds, I/O operations, and storage health.

- **2. Cross-Platform Support:**

  Operating System Compatibility: Extend compatibility to support a broader range of operating systems, ensuring users across diverse platforms can benefit from the monitoring tool.
  Mobile Device Integration: Explore possibilities for adapting the monitor to mobile platforms, enabling users to monitor and manage their systems remotely.

- **3. Enhanced Visualization and Reporting:**

  Historical Data Analysis: Implement features for storing and analyzing historical resource utilization data, allowing users to identify trends and patterns over time.
  Advanced Graphs and Charts: Introduce more sophisticated visualization tools, such as heatmaps and trend graphs, to provide deeper insights into resource usage.

- **4. Security and Anomaly Detection:**

  Security Monitoring: Integrate security-related metrics to detect and report unusual activities that could indicate security threats or unauthorized access.
  Anomaly Detection Algorithms: Implement machine learning algorithms for anomaly detection, enhancing the monitor's ability to identify irregular patterns and potential issues.

The future scope of the Resource Utilization Graph Monitor is expansive, with the potential to evolve into a versatile and indispensable tool for users seeking advanced insights, proactive system management, and collaboration in diverse computing environments.

CHAPTER 9:
REFERENCES

“Matplotlib Tutorial’’ by GeeksForGeeks
Author: Satyam Verma
Link: Article

“About the Resource Utilisation Graph” by IBM App Connect Professional.
Author: IBM
Link: Article

“Resource Allocation Graph (RAG) in Operating System” by GeeksForGeeks
Author: GeeksForGeeks
Link: Article

"What is Network Monitoring?” by WhatsUp Gold
Authors: WhatsUp Gold
Link: Article

“OS Resource Allocation Graph” by JavatPoint
Author: JavatPoint
Link: Article