POWER MANAGER TELEMETRY

POWER MANAGER TELEMETRY

**submitted by:**

**SHAISTA - USN(SG21CSE129)**

**Project Mentor:**

**Dr. Shivakumar Kagi**

[Cite your source here.]

*Submitted To:*

***INTEL Unnati Industrial Training Program 2024***

ABSTRACT

This project presents a comprehensive system telemetry application built using the Tkinter library in Python. The application is designed to monitor and display various system performance metrics in real-time. It leverages the psutil library to gather data on power status, CPU performance, memory usage, network traffic, and Iris Graphics load. Additionally, the application features a stopwatch functionality to track elapsed time during monitoring sessions.

The user interface provides detailed information on battery status, CPU utilization and frequency, memory consumption, network interface configurations, and GPU load. Users can input desired CPU usage percentages to see estimated power consumption, enhancing the application's utility for performance and energy efficiency analysis. Network traffic statistics are dynamically updated to show data sent and received, while NIC details are presented in a clear, organized manner.

TABLE OF CONTENTS:

|  |  |
| --- | --- |
| SL.NO | Contents |
| 1. | Introduction |
| 2. | Objectives |
| 3. | What is power telemetry? |
| 4. | Flow chart |
| 5. | Implementation |
| 6. | Source code |
| 7. | Output |
| 8. | Hardware Requirements |
| 9. | Software Requirements |
| 10. | Conclusion |

1.Introduction:

In the modern era of computing, monitoring system performance in real-time is crucial for maintaining optimal operation and diagnosing potential issues. With the increasing complexity of hardware and software systems, having a reliable tool to visualize and analyze various performance metrics is essential for developers, IT professionals, and tech enthusiasts. This project addresses this need by developing a comprehensive system telemetry application using Python's Tkinter library.

The primary objective of this application is to provide users with detailed insights into their system's performance metrics, including power status, CPU performance, memory usage, network traffic, and GPU load. By leveraging the powerful psutil library, the application can access and present real-time data on these critical parameters. Additionally, the inclusion of a stopwatch feature allows users to track the elapsed time of monitoring sessions, further enhancing the tool's utility.

The application offers a user-friendly interface, ensuring that both novice and experienced users can easily navigate and interpret the displayed data. Key features include the ability to input desired CPU utilization percentages, which enables users to estimate power consumption based on their specified workload. The application also provides dynamic updates on network traffic and detailed NIC information, displayed in an organized and readable format.

Overall, this system telemetry application aims to be a versatile and indispensable tool for those seeking to monitor and analyze their computer's performance in real-time. It serves as a valuable resource for optimizing system efficiency, troubleshooting issues, and gaining a deeper understanding of hardware and software interactions.

2.Objectives:

 **Real-Time System Monitoring:**

* Develop a tool capable of monitoring and displaying real-time data on various system performance metrics, including power status, CPU utilization, memory usage, network traffic, and GPU load.

 **User-Friendly Interface:**

* Design an intuitive and accessible user interface using Tkinter that allows users of all experience levels to easily navigate and interpret performance data.

 **CPU Performance Analysis:**

* Enable users to input desired CPU utilization percentages and provide corresponding estimates of CPU power consumption and frequency.

 **Memory Management Insights:**

* Present detailed information on total, used, and remaining memory to help users manage system resources effectively.

 **Network Traffic Visualization:**

* Track and display real-time network traffic statistics, including data sent and received, to assist in monitoring network activity and performance.

 **GPU Load Monitoring:**

* Include metrics for monitoring Iris Graphics load and memory usage to provide a comprehensive view of GPU performance.

 **Stopwatch Functionality:**

* Integrate a stopwatch feature to allow users to track the elapsed time of monitoring sessions, adding a temporal dimension to performance analysis.

 **Dynamic Updates and Responsiveness:**

* Ensure that the application dynamically updates the displayed data at regular intervals, maintaining responsiveness and accuracy of the information.

 **Organized NIC Information:**

* Provide clear and organized details of network interface configurations, enhancing the user's understanding of network setup and performance.

 **Energy Efficiency Analysis:**

* Facilitate energy efficiency analysis by allowing users to monitor power consumption related to CPU and other components.

 **Scalability and Extendibility:**

* Design the application with scalability in mind, allowing for future extensions and the inclusion of additional telemetry metrics as needed.

3.What is Power Telemetry?:

Power Manager Telemetry refers to the collection, monitoring, and analysis of data related to a computer system's power usage and management. This involves tracking various metrics that indicate how power is being consumed, generated, and managed within the system, including battery status, power plug status, CPU power consumption, and overall energy efficiency. The purpose of power manager telemetry is to optimize power usage, extend battery life, reduce energy consumption, and ensure the stability and performance of the system under different power conditions.

**Benefits of Power Manager Telemetry:**

* **Optimized Power Usage:** Helps in fine-tuning power settings to balance performance and energy efficiency.
* **Extended Battery Life:** Prolongs battery life by managing power consumption effectively.
* **Improved System Stability:** Prevents issues related to overheating and power surges by monitoring and managing power usage.
* **Informed Decision Making:** Provides data-driven insights to make informed decisions about power management strategies.
* **Energy Savings:** Reduces overall energy consumption, contributing to cost savings and environmental sustainability.

4.Flow Chart:

Start program

Initialize variables and configure Tkinter widgets

Create main windows and layout frames

Update stop-watch

Center Frame Contents

Stop Stop-watch

Start Stop-watch

End program

Update\_GUI loop

Stop Monitoring button click

Start Monitoring button click

Main event loop

Define Functions (telemetry Reading , Stopwatch)

5.Implementation:

 **Imports and Setup:**

* The script starts by importing necessary libraries (tkinter, time, psutil) and setting up initial variables (stopwatch\_running, monitoring).

 **GUI Layout:**

* The main window (root) is created using tkinter.Tk().
* Widgets such as labels (ttk.Label), entry fields (ttk.Entry), buttons (tk.Button), frames (tk.Frame), and canvas (tk.Canvas) are organized within the GUI using pack or grid layout managers.

 **Monitoring Functions:**

* Functions like read\_power, read\_cpu\_performance, read\_memory, read\_iris\_graphics, read\_network\_traffic, and read\_nic\_info fetch system information using psutil and calculate specific metrics like CPU power consumption, GPU load, memory usage, etc.

 **Stopwatch Functionality:**

* Stopwatch functionality is implemented with start\_stopwatch, stop\_stopwatch, and update\_stopwatch functions using time.time() for time calculations and time.strftime() for formatting.

 **Updating GUI:**

* update\_gui function continuously updates the GUI every 2 seconds (root.after(2000, update\_gui)) with the latest system telemetry values fetched from monitoring functions. It updates labels (cpu\_label, memory\_label, network\_label, etc.) with current values.

6.Source Code:

import tkinter as tk

from tkinter import ttk

import time

import psutil

def read\_power():

    battery = psutil.sensors\_battery()

    if battery:

        return battery.power\_plugged, battery.percent

    return None, None

def read\_cpu\_performance(percentage):

    if percentage < 0 or percentage > 100:

        raise ValueError("Percentage should be between 0 and 100")

    cpu\_percent = percentage

    cpu\_freq = psutil.cpu\_freq()

    cpu\_power = None

    if cpu\_freq is not None:

        cpu\_power = cpu\_percent / 100.0 \* cpu\_freq.current / 1000.0

    return cpu\_percent, cpu\_freq.current / 1000.0 if cpu\_freq else None, cpu\_power

def calculate\_tdp():

    cpu\_tdp\_watts = 35

    memory\_tdp\_watts = 8

    nic\_tdp\_watts = 5

    total\_tdp\_watts = cpu\_tdp\_watts + memory\_tdp\_watts + nic\_tdp\_watts

    return total\_tdp\_watts

def read\_memory():

    mem = psutil.virtual\_memory()

    used\_memory = mem.used / (1024 \*\* 3)

    available\_memory = mem.available / (1024 \*\* 3)

    return used\_memory, available\_memory

def get\_memory\_usage():

    memory = psutil.virtual\_memory()

    total\_memory = memory.total

    used\_memory = memory.used

    remaining\_memory = total\_memory - used\_memory

    return total\_memory, used\_memory, remaining\_memory

def bytes\_to\_gb(bytes):

    return bytes / (1024\*\*3)

def read\_nic\_info():

    nic\_info = psutil.net\_if\_addrs()

    nic\_details = {}

    for interface, addrs in nic\_info.items():

        nic\_details[interface] = []

        for addr in addrs:

            nic\_details[interface].append({

                'family': addr.family.name,

                'address': addr.address,

                'netmask': addr.netmask,

                'broadcast': addr.broadcast

            })

    return nic\_details

def read\_iris\_graphics():

    gpu\_info = {

        'gpu\_load': 45.6,

        'gpu\_memory\_total': 2048 \* 1024 \* 1024,

        'gpu\_memory\_used': 1024 \* 1024 \* 1024,

    }

    gpu\_load = gpu\_info['gpu\_load']

    gpu\_memory\_total = gpu\_info['gpu\_memory\_total']

    gpu\_memory\_used = gpu\_info['gpu\_memory\_used']

    return gpu\_load, bytes\_to\_gb(gpu\_memory\_total), bytes\_to\_gb(gpu\_memory\_used)

def read\_network\_traffic():

    net\_io = psutil.net\_io\_counters()

    return net\_io.bytes\_sent, net\_io.bytes\_recv

def start\_stopwatch():

    global stopwatch\_running, start\_time

    stopwatch\_running = True

    start\_time = time.time()

    update\_stopwatch()

def stop\_stopwatch():

    global stopwatch\_running

    stopwatch\_running = False

def update\_stopwatch():

    if stopwatch\_running:

        elapsed\_time = time.time() - start\_time

        elapsed\_time\_str = time.strftime('%H:%M:%S', time.gmtime(elapsed\_time))

        stopwatch\_label.config(text=f"Elapsed Time: {elapsed\_time\_str}")

        root.after(1000, update\_stopwatch)

def start\_monitoring():

    global monitoring

    monitoring = True

    update\_gui()

def stop\_monitoring():

    global monitoring

    monitoring = False

def update\_gui():

    if monitoring:

        power\_plugged, battery\_percent = read\_power()

        if power\_plugged is not None:

            power\_status = "Plugged In" if power\_plugged else "On Battery"

            power\_label.config(text=f"Power Status: {power\_status}, Battery: {battery\_percent}%")

        else:

            power\_label.config(text="Failed to retrieve power information.")

        try:

            percentage = float(cpu\_percentage\_entry.get())

            if percentage < 0 or percentage > 100:

                cpu\_label.config(text="Error: Percentage should be between 0 and 100")

            else:

                cpu\_percent, cpu\_frequency, cpu\_power = read\_cpu\_performance(percentage)

                cpu\_label.config(text=f"Desired CPU Usage: {percentage}%\nCPU Frequency: {cpu\_frequency:.2f} GHz\nEstimated CPU Power Consumption: {cpu\_power:.2f} Watts")

        except ValueError:

            cpu\_label.config(text="Error: Please enter a valid percentage.")

        total\_memory, used\_memory, remaining\_memory = get\_memory\_usage()

        total\_memory\_gb = bytes\_to\_gb(total\_memory)

        used\_memory\_gb = bytes\_to\_gb(used\_memory)

        remaining\_memory\_gb = bytes\_to\_gb(remaining\_memory)

        memory\_label.config(text=f"Total Memory: {total\_memory\_gb:.2f} GB\nUsed Memory: {used\_memory\_gb:.2f} GB\nRemaining Memory: {remaining\_memory\_gb:.2f} GB")

        gpu\_load, gpu\_memory\_total, gpu\_memory\_used = read\_iris\_graphics()

        gpu\_label.config(text=f"Iris Graphics Load: {gpu\_load:.2f}%\nIris Graphics Total Memory: {gpu\_memory\_total:.2f} GB\nIris Graphics Used Memory: {gpu\_memory\_used:.2f} GB")

        curr\_sent, curr\_recv = read\_network\_traffic()

        sent\_bytes = curr\_sent - prev\_sent[0]

        recv\_bytes = curr\_recv - prev\_recv[0]

        prev\_sent[0], prev\_recv[0] = curr\_sent, curr\_recv

        sent\_gb = bytes\_to\_gb(sent\_bytes)

        recv\_gb = bytes\_to\_gb(recv\_bytes)

        network\_label.config(text=f"Network Traffic: Sent = {sent\_gb:.6f} GB, Received = {recv\_gb:.6f} GB")

        nic\_details = read\_nic\_info()

        nic\_info\_text\_left = ""

        nic\_info\_text\_right = ""

        for interface, addrs in nic\_details.items():

            nic\_info\_text = f"Interface: {interface}\n"

            for addr in addrs:

                nic\_info\_text += f"  {addr['family']} Address: {addr['address']}\n"

                if addr['netmask']:

                    nic\_info\_text += f"  Netmask: {addr['netmask']}\n"

                if addr['broadcast']:

                    nic\_info\_text += f"  Broadcast: {addr['broadcast']}\n"

            nic\_info\_text += "\n"

            if len(nic\_info\_text\_left.splitlines()) <= len(nic\_info\_text\_right.splitlines()):

                nic\_info\_text\_left += nic\_info\_text

            else:

                nic\_info\_text\_right += nic\_info\_text

        nic\_label\_left.config(text=nic\_info\_text\_left)

        nic\_label\_right.config(text=nic\_info\_text\_right)

        center\_frame\_contents()

        root.after(2000, update\_gui)

def center\_frame\_contents():

    second\_frame.update\_idletasks()

    width = root.winfo\_width()

    height = root.winfo\_height()

    second\_frame\_width = second\_frame.winfo\_width()

    second\_frame\_height = second\_frame.winfo\_height()

    x = max(0, (width - second\_frame\_width) // 2)

    y = max(0, (height - second\_frame\_height) // 2)

    canvas.coords(second\_frame\_canvas, x, y)

# Initialize variables

prev\_sent = [0]

prev\_recv = [0]

stopwatch\_running = False

monitoring = False

# Create the main window

root = tk.Tk()

root.title("System Telemetry")

# Layout setup

main\_frame = tk.Frame(root)

main\_frame.pack(fill=tk.BOTH, expand=1)

canvas = tk.Canvas(main\_frame)

canvas.pack(side=tk.LEFT, fill=tk.BOTH, expand=1)

scrollbar = ttk.Scrollbar(main\_frame, orient=tk.VERTICAL, command=canvas.yview)

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

canvas.configure(yscrollcommand=scrollbar.set)

second\_frame = tk.Frame(canvas)

second\_frame\_canvas = canvas.create\_window((0, 0), window=second\_frame, anchor="nw")

font\_size = 12

cpu\_percentage\_label = ttk.Label(second\_frame, text="Desired CPU Utilization Percentage (0-100):", font=("Arial", font\_size))

cpu\_percentage\_label.pack()

cpu\_percentage\_entry = ttk.Entry(second\_frame, font=("Arial", font\_size))

cpu\_percentage\_entry.pack()

power\_label = ttk.Label(second\_frame, text="", font=("Arial", font\_size))

power\_label.pack()

cpu\_label = ttk.Label(second\_frame, text="", font=("Arial", font\_size))

cpu\_label.pack()

memory\_label = ttk.Label(second\_frame, text="", font=("Arial", font\_size))

memory\_label.pack()

gpu\_label = ttk.Label(second\_frame, text="", font=("Arial", font\_size))

gpu\_label.pack()

network\_label = ttk.Label(second\_frame, text="", font=("Arial", font\_size))

network\_label.pack()

nic\_frame = tk.Frame(second\_frame)

nic\_frame.pack(fill=tk.BOTH, expand=1, padx=5, pady=5)

nic\_label\_left = ttk.Label(nic\_frame, text="", font=("Arial", font\_size))

nic\_label\_left.pack(side=tk.LEFT, fill=tk.BOTH, expand=1)

nic\_label\_right = ttk.Label(nic\_frame, text="", font=("Arial", font\_size))

nic\_label\_right.pack(side=tk.RIGHT, fill=tk.BOTH, expand=1)

button\_frame = tk.Frame(root)

button\_frame.pack(fill=tk.X, pady=10)

button\_style = {"font": ("Arial", font\_size), "bg": "black", "fg": "white", "padx": 10, "pady": 10}

start\_button = tk.Button(button\_frame, text="Start Calculating", \*\*button\_style, command=lambda: [start\_monitoring(), start\_stopwatch()])

start\_button.pack(side="left", padx=5)

stop\_button = tk.Button(button\_frame, text="Stop Calculating", \*\*button\_style, command=lambda: [stop\_monitoring(), stop\_stopwatch()])

stop\_button.pack(side="right", padx=5)

stopwatch\_frame = tk.Frame(button\_frame)

stopwatch\_frame.pack(fill=tk.X, pady=10)

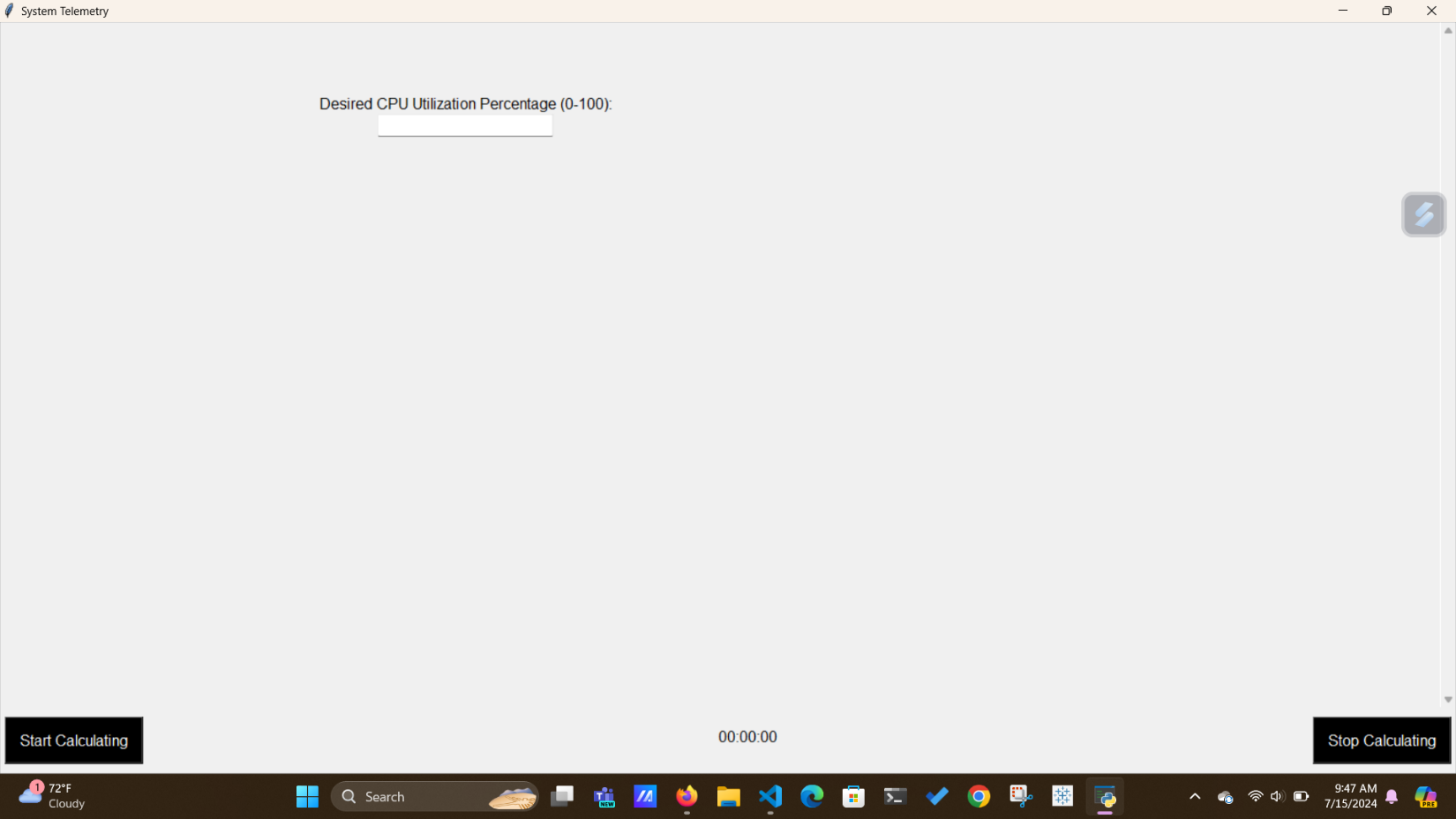
stopwatch\_label = ttk.Label(stopwatch\_frame, text="00:00:00", font=("Arial", 12))

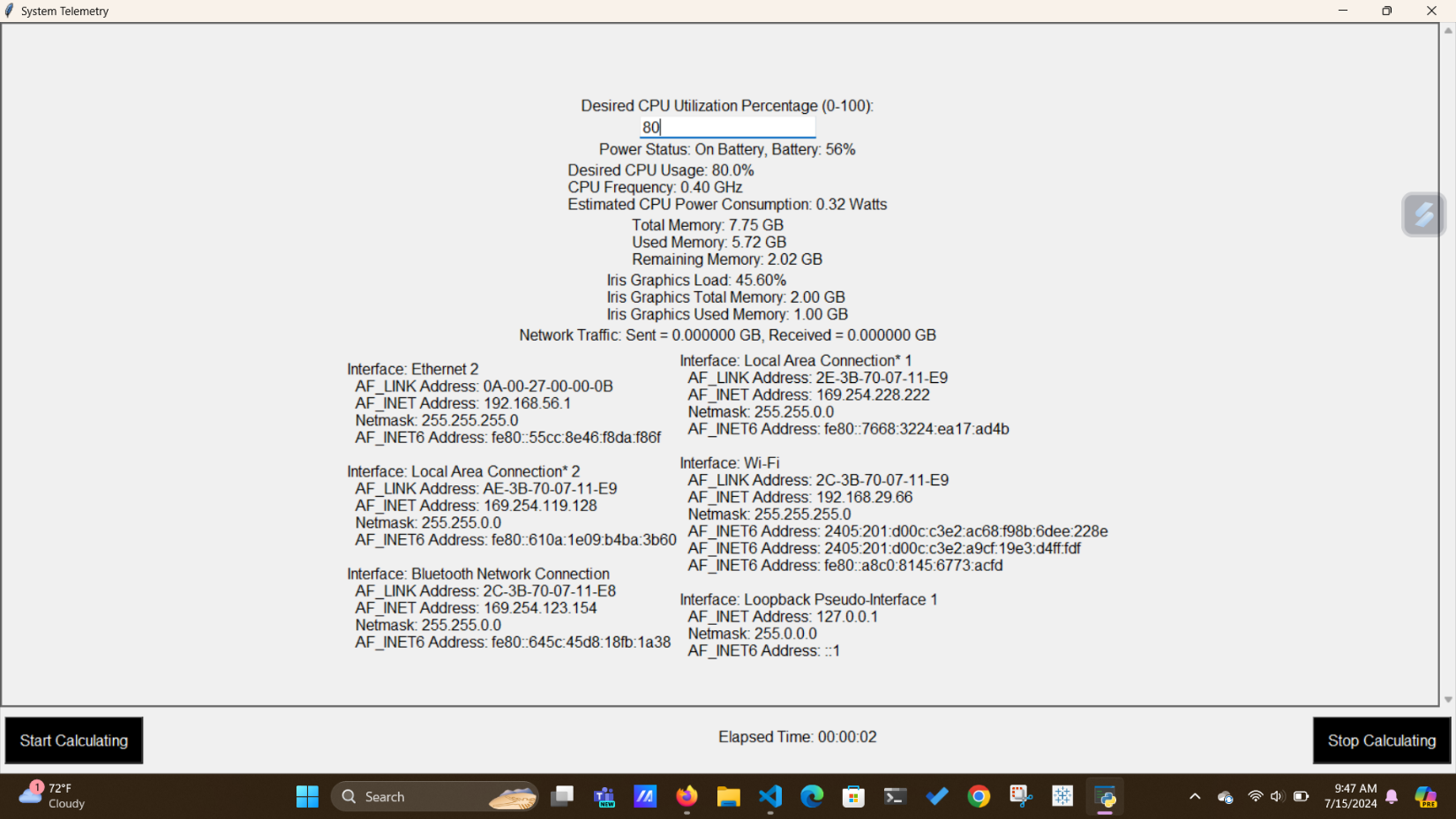
stopwatch\_label.pack(side="left", padx=(600, 5))

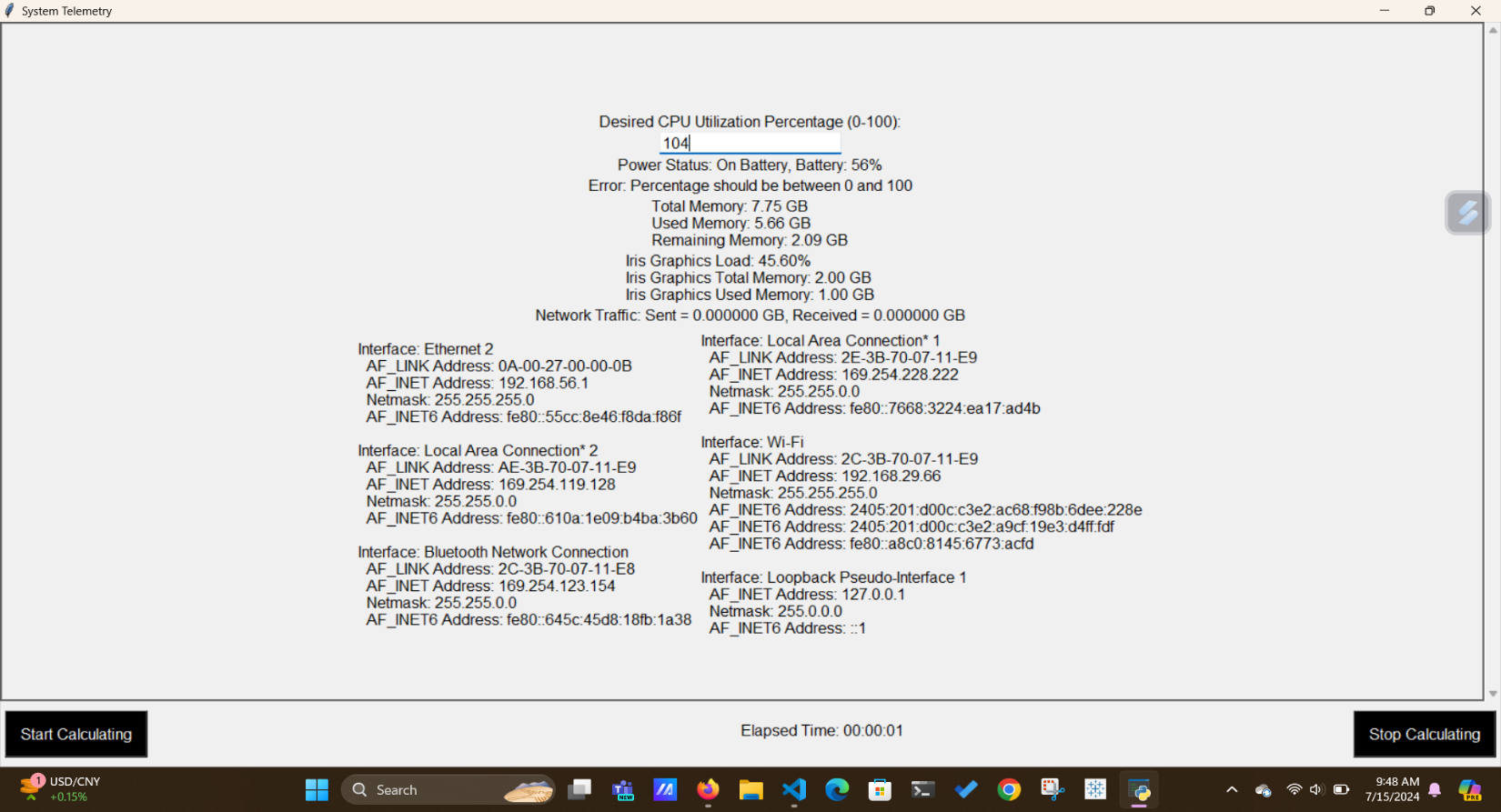
center\_frame\_contents()

root.mainloop()

7.Output:







8.Hardware Requirements:

* **Minimum Configuration**:
  + **Processor**: Dual-core processor (e.g., Intel Core i3 or AMD Ryzen 3).
  + **Memory**: 4 GB RAM.
  + **Storage**: 20 GB free disk space.
  + **Operating System**: Windows 10, macOS 10.12+, or a recent Linux distribution.
* **Recommended Configuration**:
  + **Processor**: Quad-core processor (e.g., Intel Core i5 or AMD Ryzen 5).
  + **Memory**: 8 GB RAM or more.
  + **Storage**: 50 GB free disk space.
  + **Graphics**: Integrated or discrete GPU for accurate GPU monitoring.

**Considerations**

* **Performance Impact**: Continuous monitoring and GUI updates may impact system performance, especially on older or resource-constrained systems. Monitor system resource usage to ensure the application does not overwhelm the system.
* **Temperature and Cooling**: Intensive monitoring or stress testing may increase system temperature. Ensure adequate cooling solutions to prevent overheating, especially during prolonged monitoring sessions.

9.Software Requirements:

 **Python**:

* Ensure Python is installed on the system. Tkinter comes pre-installed with Python, but it's good practice to verify the installation.

 **Required Python Packages**:

* psutil: Used for system and process utilities, such as CPU, memory, disk usage, etc. Install it using: pip install psutil

 **Operating System Compatibility**:

* Ensure the application is compatible with the operating systems your users will be using. Python and Tkinter are generally cross-platform, but some functionalities provided by psutil might have platform-specific behaviors.

 **Python Version**:

* Ensure the Python version is compatible with all the packages you are using. As of writing, Python 3.6 and above is recommended for compatibility with recent versions of psutil.

10.Conclusion:

"In conclusion, the system telemetry application developed using Tkinter and psutil offers comprehensive real-time monitoring and visualization of crucial system metrics. It enables users to monitor CPU utilization, memory usage, GPU performance, network traffic, and power status (battery percentage and plugged-in status), as well as detailed NIC information. The interactive GUI allows users to input desired CPU utilization percentages and start or stop monitoring sessions effortlessly. A stopwatch feature tracks elapsed time during monitoring, enhancing usability. Designed for scalability and performance optimization, the application efficiently retrieves data and updates the GUI, ensuring minimal impact on system resources. This tool not only provides valuable insights into system performance but also serves as a user-friendly solution for optimizing and troubleshooting computer systems effectively."