**SIMPLE HARDWARE RESOURCE VIEWER**

**Group 10: Taylor DeMarche**

**Fall 2024**

***Table of Contents:***

Abstract 3

Work distribution 3

Introduction 3

Methodology 3

Implementation 4

Results and Discussion 6

Challenges and Lessons Learned 7

Future Work 8

References: 9

Appendices: 9

***List of Figures:***

[Figure 1: Main Interface on Startup. 4](#_Toc184238476)

[Figure 2: Regular Expression Usage. 5](#_Toc184238477)

[Figure 3: Log Frame Visible. 5](#_Toc184238478)

[Figure 4: Show Log Button Implementation. 6](#_Toc184238479)

[Figure 5: Network Data Gathering. 7](#_Toc184238480)

# Abstract

This project was my attempt to create a hardware resource interface. The goal was to create the interface itself as well as all of the background functionality that will collect the usage data for individual components of the machine and present them to the user with a minimalist approach. There were some struggles along the way as I encountered bugs and interesting challenges that were overcome during the process of completing the final stages. The impact of this project has left me with multiple ideas of additional work and extra features that I’d like to tackle in the future.

# Work distribution

As the sole member of my group, I take full responsibility for the outcome of this project, the written portion as well as the presentation and the code.

# Introduction

The goal of this project was to address topic 11 and create an interface that displays hardware performance information. My personal objective was to build it from the ground up and make it very user-friendly while including the most relevant information. On top of that I wanted the information given to be accurate and concise without any extra information that isn’t necessarily helpful to users.

# Methodology

I chose Python as the programming language due to my familiarity with it, and it’s approachable learning curve. It also means that the program can run on any operating system that is compatible with Python 3. I’ve implemented several modules, such as tkinter, psutil, platform, and others. The plan was to present information that is easy to read at a glance, have it update periodically, and create an optional log of the hardware usage.

Two similar projects would be “[Glance: An Eye on your system](https://nicolargo.github.io/glances/)” and “[Netdata](https://www.netdata.cloud/)”. They have graphs showing different information regarding the usage of hardware as well as analyses such as showing the amount of entropy in the data.

The software that I used to create this project was a standard installation of Python 3.12, as well as the Spyder IDE. Spyder is one of the more preferred Python IDE’s as you can execute and debug code as well as see variables and other information in real time. I have previously used this software for other classes, but I referred to the Spyder documentation at one point as I ran into an issue. I was quitting the interface window, but the kernel was not aborting the kernel process, which made subsequent tests unable to initiate and wasn’t exiting the process correctly. To correct this I had to call both functions *tkinter.TK.destroy()* and *sys.exit()* during the quit sequence when the quit button was pressed.

# Implementation

A screenshot of a computer

Description automatically generated

#### Figure 1: Main Interface on Startup.

This is the main window that appears when the program is started. The interface was developed using tkinter including labels, messages, text, and buttons. I created 4 separate columns using the grid functionality. Each is assigned to a type of hardware present in all computers. From left to right I have processor (CPU) data, RAM, disk/SSD memory, and network usage.

In the first column, the processor information is displayed, including the brand, model number, number of cores, and maximum speed. The total installed RAM memory is on display in the next column. Current total hard drive or SSD memory installed (not including the OS partition) and the amount used in terabytes are displayed within the third column. Then in the final fourth column it indicates the network type, either Wi-Fi or ethernet.

The processor information was a little bit more challenging to gather as the information can vary between systems. To make sure I got the accurate information, I implemented a couple regular expressions. One (m1) that matches the beginning section of the string that is returned along with a number. This match accounts for “AMD64” and would accommodate for other processors or that might be installed. The second match searches for the model number, which can be an integer or a decimal number. These expressions are shown in the snippet of code in the screenshot here.

A computer screen shot of a computer screen

Description automatically generated

#### Figure 2: Regular Expression Usage.

The current data section is across the middle row and color coded to correspond with the labels above. It shows the available amount of unused capability and the percentage of what is being utilized for each hardware section. The network information shown in this area is the raw sent bytes and received bytes per second (KB) over the previous five seconds during the program’s runtime.

A green button is located on the bottom left with the text “Show Log” on it. This button will show the usage logs that were generated beginning at the start of the program, updating with a new line every five seconds. When the amount of data overloads the size that the window allows for the information will acquire a scrollbar to navigate down the lists.

A screenshot of a computer

Description automatically generated

#### Figure 3: Log Frame Visible.

The usage logs show the information from the sections in color in a shorthand representation (A = Available, U = Used) and every time they are updated the new information is appended to the bottom of their respective lists. The “Show Log” button is also switched to a “Close Log” button. This means that clicking the same button that showed the usage log area will remove the usage logs from the screen when clicked.

A computer screen shot of a program code

Description automatically generated

#### Figure 4: Show Log Button Implementation.

The functions that dictate that button are shown in this snippet of code. Because tkinter allows for a grid layout to place each item on the window, a function is called to place the frame that the logs are displayed in onto the grid. Then it configures the button to now contain the command for the *remove\_log\_frame* function. That line sets the button so that the next click will remove the usage log from the grid which hides the information. Once the data is gone it will re-assign the button back to the original *show\_log\_frame* function so that the usage log can be toggled on and off.

The “Show Log” and “Quit” button are both set to remain in place when the window is resized. The tkinter attribute for this functionality is called sticky. It allows the object to be assigned to a cardinal direction (sw and se respectively) within its section of the grid. This functionality, along with the grid being configured to have weights that determine the stretch factor allows for the window to be maximized and fitted to any size screen without losing the overall layout.

The main window that was created has a function called *mainloop* that starts the event handling to check if buttons are pressed. It manages the interactions for me which is why tkinter is fairly easy to use. The function that actually populates the data is a user-defined function called *update\_log*. I pull all the data and then after some math to make sure it’s represented in an easily read format (e.g., displaying terabytes with two decimal places instead of the number of bytes). The function then configures the corresponding message objects in the color-coded row with the new data. Then appends the same data to the text objects in the usage logs.

# Results and Discussion

I feel as though the end design is what I set out to achieve in that it is very minimal and easy to view. It has the data usage correctly showing and the logs are accurate. When I ramped up usage on my machines to test it the values spiked as appropriate, and it was the same to that of the task manager values. The functionality is somewhat lacking as being alone on this I didn’t have as many features added that I would expect from a project with multiple people compounding on the time spent to flush things out. That being said, everything that was intended to get this project started is complete and accurate. Given more time I will be returning to this to get more working parts and enhance each piece to be even better.

# Challenges and Lessons Learned

I came across a couple of moments like the quit button where bugs arose during development. None that proved to be insurmountable fortunately. The most significant one came up when setting up the disk memory section of the code. When I attempted to gather the information regarding memory disk usage, I found that there was a function that returns the data I would need built into the psutil module. When I attempted to use this function (*psutil.disk\_usage(‘/’)*) there was an error that was stopping the program during runtime.

After some research I found out that there is a bug with Python 3.12 and this particular function. I found a link to a GitHub project pull request, “*Fix for issue #2310* [Pull request #2310],” where the developer implemented another module that has a similar function called shutil. This module has a function by the same name that is given the same parameters, however this function is compatible with version 3.12 of Python.

Another issue that proved to be a puzzle that I had to work around was the network data. I was able to find a function from psutil again that would create a dictionary of all the different types of internet connections and includes the raw data of bytes sent and received.

A computer screen with text

Description automatically generated

#### Figure 5: Network Data Gathering.

When I first used this to set the sent and received information for each data type, I assumed that the numbers were correct as it showed the current connection type had data, and it would be raised between function calls. The problem was that I wanted the information between intervals only to be displayed in my project. To that effect, I found a function that will clear the cache of data and it should have made each successive call show the network data that was sent/received since the previous update. This however didn’t work. I learned that on certain versions of the module that function doesn’t clear the cache to zero, but instead prevents the data from wrapping around to zero once the buffer overflows.

In order to fix this problem I had to get a little creative. Initializing each of the network data variables and declaring them as global variables within the update function allows me to retain the old data upon the next iteration of the updates. I opted to use temp variables to hold the old data during the update, pull the new data into the global variables, and then subtract the old from the new. The difference is divided by the interval length to get the average network data usage per second. Using a string to denote the connection type in use I make sure to update the text in the log and the current data sections using the correct corresponding information whether it is ethernet or wi-fi.

# Future Work

If I am able to have more time and resources for this project in the future I’d like to add a few features and flex my creativity to create something more eye catching and pretty to look at. As of right now I just have slightly different shades of blue to not be too flashy and distracting while still hinting at the correspondence. As well as different shades of muted gray backgrounds to maintain the readability. Being able to explore more tkinter options to ensure everything is perfectly aligned would be an improvement. Maybe adding some images or flair to the data shown for a more vibrant or robust aesthetic would also enhance the overall feel of the project.

One of the features I’d like to implement would be another button that opens a second window or another section where the applications that are currently running would be shown with a breakdown of their specific hardware usage. Multiple organizations of the list would ideally be available. Like showing the background processes separately from the user enabled/interactive applications or sort by name or time the process was started. Maybe even including a filter or search bar so that the information is streamlined.

A second feature that would be great to implement exists in the two other example projects I mentioned earlier that are similar. Namely the presence of graphical data. Even if the graphs only show the data pulled during the update interval, having that visual effect is very helpful to gain knowledge at a glance which is a main goal of the project.

Also, implementing a drop down menu or some other kind of user input to allow the user to determine the interval they would like for updates would be substantial. This information could be stored in a config file for the application and when it is changed the program will remember that for the next time it is run. Among other things like what filters were chosen for the application data, it would be very helpful to have settings that are retained between sessions of the program.

In conclusion, developing my Simple Hardware Resource Viewer has been engaging and quite educational. While challenging at times, it was a very rewarding experience to have the project come together piece by piece with a full understanding of each part. Initially I regretted working alone. However, now that I have a unique project where I can feel comfortable modifying anything and everything, I’m grateful and excited to learn how to do more and more in my future career.

# References:

1. Python Software Foundation. (n.d.). *Tkinter — Python interface to Tcl/Tk*. Retrieved October 2024, from <https://docs.python.org/3/library/tkinter.html>
2. TkDocs. (n.d.). *Grid geometry management*. Retrieved October 2024, from <https://tkdocs.com/tutorial/grid.html>
3. Giampaolo. (2020, July 31). *Fix for issue #2310* [Pull request #2310]. GitHub. Retrieved October 2024, from <https://github.com/giampaolo/psutil/pull/2310/files>
4. Psutil. (n.d.). *Psutil documentation*. Retrieved October 2024, from <https://psutil.readthedocs.io/en/latest/index.html>
5. Python Software Foundation. (n.d.). *Platform — The platform module*. Retrieved October 2024, from <https://docs.python.org/3/library/platform.html>

# Appendices:

1. Spyder IDE. (n.d.). *Spyder documentation*. Retrieved October 2024, from <https://docs.spyder-ide.org/current/index.html>