

Deep Dive : A System Resource Monitor

Seminar (IT290) Report

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

In

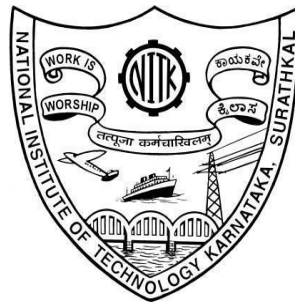
INFORMATION TECHNOLOGY

by

NITHIN S 221IT085

AYUSH KUMAR 221IT015

JAY CHAVAN 221IT020



DEPARTMENT OF INFORMATION TECHNOLOGY
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALORE -575025

March, 2024

DECLARATION

We hereby *declare* that the *Seminar (IT290) Report* entitled “**Deep Dive : A System Resource Monitor**” which is being submitted to the National Institute of Technology Karnataka Surathkal, in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in the Department of Information Technology, is a *bonafide report of the work carried out by us*. The material contained in this project report has not been submitted to any University or Institution for the award of any degree.

Nithin S

Ayush Kumar

Jay Chavan

Signature
Department of IT

Signature
Department of IT

Signature
Department of IT

Place : NITK, SURATHKAL
Date : 29 March 2024

CERTIFICATE

This is to certify that the Seminar entitled “**Deep Dive : A System Resource Monitor**” has been presented by Nithin S (221IT085), Ayush Kumar (221IT015) & Jay Chavan (221IT020), students of IV semester B.Tech.(I.T), Department of Information Technology, National Institute of Technology Karnataka, Surathkal, on 29 March, 2024, during the even semester of the academic year 2023 - 2024, in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Information Technology.

Examiner-1 Name

Signature of the Examiner-1 with Date

Examiner-2 Name

Signature of the Examiner-2 with Date

Place:

Date:

ABSTRACT

This report presents Deep Dive, a System Resource Monitor which is a sophisticated software project aimed at providing users with an intuitive and comprehensive tool for monitoring system resources in real-time. Leveraging the power of Python and Qt GUI interface, this project offers a user-friendly experience while ensuring accurate tracking and analysis of critical system metrics. The system resource monitor provides insights into CPU usage, memory consumption, disk activity, network traffic, and other vital parameters, enabling users to identify performance bottlenecks, optimize resource utilization, and troubleshoot potential issues efficiently. With its responsive and visually appealing interface, Deep Dive empowers both novice and advanced users to make informed decisions about system resource management, thereby enhancing system stability, reliability, and overall performance.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION.....	1
CHAPTER 2: TECHNICAL DISCUSSION.....	3
Importance in Real-World Application.....	3
Relevant Applications with Examples.....	4
Literature Review.....	4
CHAPTER 3: WORK DONE.....	5
Code Structure.....	5
Block Diagram.....	7
Origin of Statistics.....	8
UI Design.....	14
Results.....	15
CHAPTER 4: CONCLUSION AND FUTURE TRENDS.....	20
Conclusion.....	20
Future Work.....	20
References.....	21

LIST OF FIGURES

Figure 3.1: Terminal Activity of the application.....	6
Figure 3.2: Flowchart of Application Architecture.....	7
Figure 3.3: UI design in Qt Designer.....	14
Figure 3.4: Assets Tab Screen.....	15
Figure 3.5: Processes Tab Screen.....	16
Figure 3.6: Process Minitab Screen.....	16
Figure 3.7: Performance Tab Screen.....	17
Figure 3.8: Connections Tab Screen.....	18
Figure 3.9: Partition Tab Screen.....	19
Figure 3.10: Information Tab Screen.....	19
Figure 3.11: Project Details Tab Screen.....	19

LIST OF TABLES

Table 3.1: Origin of Statistics.....	8
--------------------------------------	---

CHAPTER 1: INTRODUCTION

In the dynamic realm of modern computing, the efficient management of system resources emerges as a critical necessity, underpinning the smooth operation and longevity of computing environments across diverse domains. As the complexity of software applications escalates and the demands on hardware intensify, the need for sophisticated tools that facilitate real-time monitoring and analysis of system resources becomes increasingly pronounced. It is within this context that our project, "Deep Dive: A System Resource Monitor," assumes significance. Deep Dive represents a concerted effort to address the multifaceted challenges associated with system resource management by providing users with a comprehensive and intuitive platform for monitoring and analyzing crucial system metrics. By harnessing the capabilities of Python for robust backend functionality and Qt GUI interface for seamless user interaction, Deep Dive endeavors to redefine the paradigm of system resource monitoring, offering users unprecedented insights into CPU usage, memory consumption, disk activity, network traffic, and other key performance indicators.

Language Used : Python

Libraries Used : Psutil, Os, PyQt5, PyCPUInfo, Distro, Numpy

Working Environment : Any Linux Distro

Requirements : A Linux Environment, Numpy, Psutil, Distro, PyQt5, PyQtGraph, PyCPUinfo, PyQtChart

At the core of Deep Dive lies a steadfast commitment to empowering users with the tools and insights necessary to optimize system resources effectively. Our project sets out with a singular objective: to equip users with the means to not merely monitor but also proactively manage system resources in real-time. Through the provision of actionable insights into critical performance metrics, Deep Dive enables users to identify bottlenecks, fine-tune resource allocation, and swiftly address any arising performance issues. By fostering a deeper understanding of system resource utilization patterns and trends, the tool aims to enhance system efficiency, reliability, and overall performance. Through its intuitive interface and advanced

visualization techniques, Deep Dive seeks to democratize system resource management, making it accessible and comprehensible to users across diverse technical backgrounds and proficiency levels.

The objective is to explore all these fields:

- **CPU:** Percentage, Clock Speeds, Occupancy, Duration, Counts of Context Switches, System calls and hardware interrupts
- **Memory:** Utilization, Swapping rate, Occupancy, Physical and logical partition.
- **Internet:** Packet Count, Size, Rate, Transfer rate, Transmission, Dropped transfers.
- **Processes:** PIDs, names, terminal, usernames, states, CPU/Memory Usage, context switches, threads, Kill, resume, Terminate, Suspend Options.
- **System/Hardware:** OS, Kernel names, Versions, CPU name, Vendor, Frequency, Features.
- Visually represent real time data changes for clear understanding

The main goal is to integrate all these metrics into one application and give a naive user all the in depth details about his/her system, which is not done by conventional system monitors.

CHAPTER 2: TECHNICAL DISCUSSION

In today's rapidly advancing technological landscape, the efficient monitoring of system resources is paramount for ensuring optimal performance, stability, and security in various computing environments. System resource monitoring encompasses the tracking and analysis of crucial components such as CPU utilization, memory usage, disk activity, network traffic, and application performance. This chapter delves into the importance of system resource monitoring in real-world applications, relevant examples, and a literature review to underscore its significance and practical implications.

Importance in Real-World Applications

System resource monitoring serves as the backbone of IT infrastructure management, ensuring the smooth operation of systems and applications. In enterprise environments, the ability to monitor CPU, memory, disk, and network usage in real-time is crucial for maintaining uptime, optimizing performance, and identifying potential issues before they escalate into critical problems.

For example, in e-commerce platforms, sudden spikes in CPU utilization or disk I/O could indicate increased user activity or a surge in transactions. Without effective monitoring, such events could lead to performance degradation or system failures, resulting in lost revenue and customer dissatisfaction. By implementing robust resource monitoring solutions, businesses can proactively respond to changing demands, scale resources accordingly, and deliver a seamless user experience.

In cloud computing, where resources are provisioned dynamically, monitoring becomes even more essential. Cloud providers offer monitoring services that enable users to track resource utilization, set alarms for predefined thresholds, and automate scaling based on demand. This ensures optimal resource allocation, cost efficiency, and adherence to SLAs.

Moreover, system resource monitoring plays a crucial role in cybersecurity by detecting anomalous behavior and potential security threats. By analyzing system logs, network traffic patterns, and application performance metrics, security teams can identify indicators of compromise, unauthorized access attempts, or malicious activities. Timely detection and response are vital for mitigating security breaches and safeguarding sensitive data.

Relevant Applications with Examples

Network Operations Centers (NOCs): In NOCs, teams monitor the health and performance of network devices, servers, and applications using specialized monitoring tools such as SolarWinds or PRTG Network Monitor. These tools provide real-time insights into network traffic, bandwidth utilization, and device status, allowing administrators to troubleshoot issues and optimize network performance.

Healthcare IT Systems: Hospitals and healthcare organizations rely on system resource monitoring to ensure the availability and reliability of critical healthcare applications such as Electronic Health Records (EHR) systems and Picture Archiving and Communication Systems (PACS). Monitoring tools such as Splunk or Dynatrace help IT teams monitor application performance, database responsiveness, and server health to deliver uninterrupted patient care services.

Financial Services: In the financial sector, where high-frequency trading and real-time transaction processing are paramount, system resource monitoring is essential for maintaining low latency and high availability. Trading firms utilize monitoring tools like Datadog or New Relic to monitor market data feeds, trading algorithms, and infrastructure performance to execute trades swiftly and accurately.

Literature Review

Several research studies have contributed to advancing the field of system resource monitoring by proposing novel techniques, algorithms, and frameworks to address emerging challenges. For instance, research by Li et al. (2021) introduces a machine learning-based anomaly detection approach for identifying performance anomalies in cloud environments, improving detection accuracy and reducing false positives.

Furthermore, the work of Sharma et al. (2019) explores the application of containerization and microservices architecture in system monitoring, enabling more granular monitoring of application components and efficient resource utilization in cloud-native environments.

In the realm of edge computing and IoT, research by Yang et al. (2020) presents a lightweight monitoring framework for resource-constrained edge devices, enabling efficient resource management and workload distribution in edge computing networks.

CHAPTER 3: WORK DONE

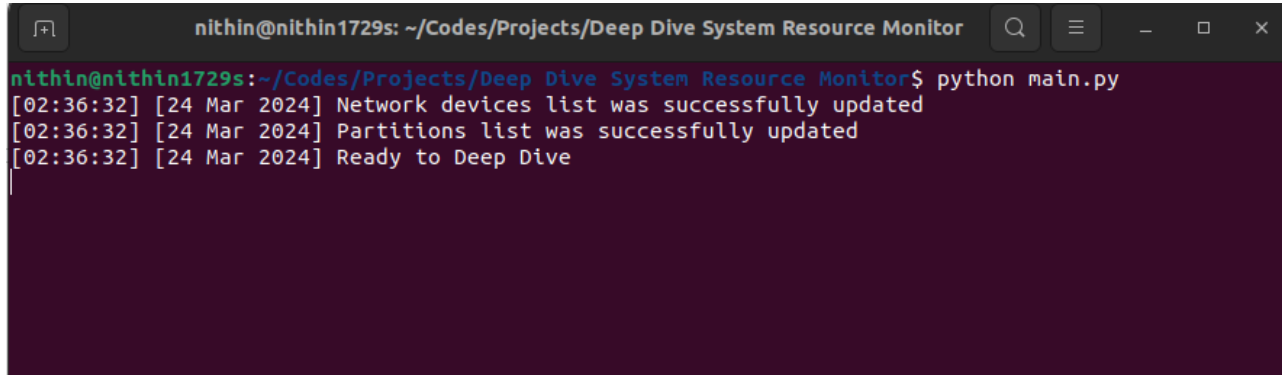
The development of Deep Dive is underpinned by a meticulously crafted methodology that combines rigorous data collection, sophisticated analysis, and intuitive presentation of insights. Leveraging Python's versatility and robustness, the tool employs a multifaceted approach to gather, process, and analyze data pertaining to various system metrics, including CPU usage, memory utilization, disk activity, network statistics, and process details. Through the adept utilization of algorithms and techniques, Deep Dive ensures the accuracy, responsiveness, and reliability of resource monitoring, even in the face of dynamic and rapidly changing computing environments. Real-time visualization of this data constitutes a cornerstone of the tool, facilitating instantaneous insights into system performance and enabling users to make informed decisions regarding resource management. By adhering to a structured methodology that prioritizes user-centric design principles and robust backend functionality, Deep Dive endeavors to deliver a seamless and immersive experience that empowers users to unlock the full potential of their computing environments.

Code Structure

1. **commons:** Consist for python code for the frontend of the application. Its designed using Qt5 Designer
2. **readers:**
 - **mainwind** : Stands for the main window. The data for all 6 tabs are collected here.
 - **procwind** : Stands for the process window. It provides user interaction for process control.
3. **screens:**
 - **mainwind** : Consists of interface design for all the tabs.
 - **procwind** : Consists of interface design for the mini process tabs.
4. **widgets :**
 - **lgptwdgt, ntwkwdgt, perfwdgt, phptwdgt** : All these are used to design widgets for tabs

5. ui files :

- Consists of ui files made in Qt Designer for all the tabs.

A terminal window titled 'nithin@nithin1729s: ~/Codes/Projects/Deep Dive System Resource Monitor'. The terminal shows the command 'python main.py' being executed. The output consists of three lines of log messages: '[02:36:32] [24 Mar 2024] Network devices list was successfully updated', '[02:36:32] [24 Mar 2024] Partitions list was successfully updated', and '[02:36:32] [24 Mar 2024] Ready to Deep Dive'. The terminal has a dark background with light-colored text.

```
nithin@nithin1729s: ~/Codes/Projects/Deep Dive System Resource Monitor$ python main.py
[02:36:32] [24 Mar 2024] Network devices list was successfully updated
[02:36:32] [24 Mar 2024] Partitions list was successfully updated
[02:36:32] [24 Mar 2024] Ready to Deep Dive
```

Figure 3.1: Terminal Activity of the application

Deep Dive employs a variety of techniques for data collection to ensure comprehensive coverage of system metrics. This includes interfacing with operating system APIs to retrieve real-time data on CPU usage, memory utilization, disk activity, network statistics, and process details. Additionally, the tool may utilize performance monitoring tools or libraries to access low-level system information with precision and accuracy.

Deep Dive prioritizes real-time visualization as a key component of its user interface. Utilizing modern visualization libraries and techniques, the tool presents system metrics in a visually appealing and intuitive manner. This allows users to quickly grasp the current state of their system's performance and identify any areas requiring attention. Interactive elements may be incorporated to enable users to drill down into specific metrics or timeframes for deeper analysis. Deep Dive is designed to adapt seamlessly to dynamic and rapidly changing computing environments. This adaptability is achieved through robust error handling, graceful degradation, and automatic recovery mechanisms to maintain uninterrupted monitoring even in the face of transient errors or system fluctuation

Block Diagram

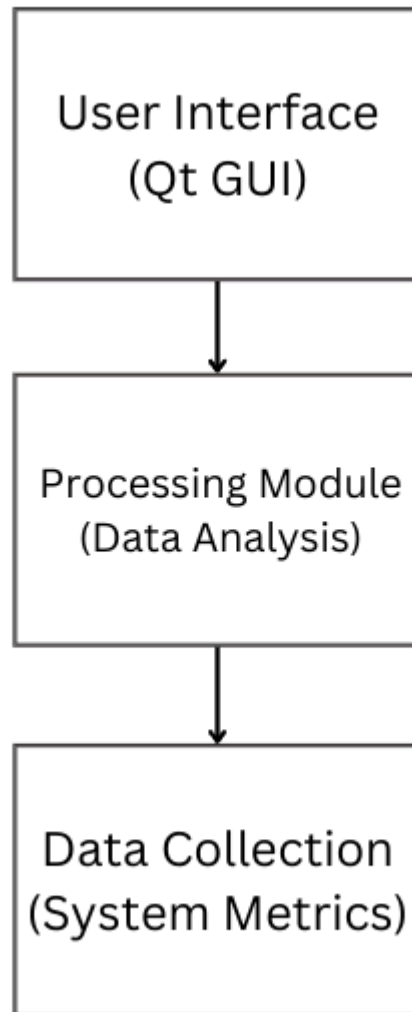


Figure 3.2: Flowchart of Application Architecture

The Data Collection is done by Python libraries like Psutil, PyCPUInfo, Distro and os. These raw system metrics are analyzed to prepare meaningful statistics in the processing module. More information regarding this is given in the next chapter.

These processed stats are displayed in a user-friendly manner by the PyQt interface.

Origin of Statistics

The below table shows how the various statistics are collected by our System Resource Monitor using Python Libraries.

Description	Command Used
Name of the Linux Distro	Distro.name()
Version of the Linux Distro	distro.version()
Hostname of the system	uname().nodename
Release of the OS	uname().version
Version information of the operating system	uname().version
Boot time of the system	psutil.boot_time()
Bytes received	nxntioctf.bytes_recv - pvntioctf.bytes_recv
Bytes sent	nxntioctf.bytes_sent - pvntioctf.bytes_sent
Packets received	nxntioctf.packets_recv - pvntioctf.packets_recv
Packets sent	nxntioctf.packets_sent - pvntioctf.packets_sent
Bytes received per network interface	nxntioct[indx].bytes_recv - pvntioct[indx].bytes_recv
Bytes sent per network interface	nxntioct[indx].bytes_sent - pvntioct[indx].bytes_sent
Packets received per network interface	nxntioct[indx].packets_recv - pvntioct[indx].packets_recv
Packets sent per network interface	nxntioct[indx].packets_sent - pvntioct[indx].packets_sent
Total bytes received	netiocnf.bytes_recv
Total bytes sent	netiocnf.bytes_sent
Total packets received	netiocnf.packets_recv

Total packets sent	netiocnf.packets_sent
Receive errors	netiocnf.errin
Send errors	netiocnf.errout
Dropped packets on receive	netiocnf.dropin
Dropped packets on send	netiocnf.dropout
Disk read count	diskioqt.read_count
Disk write count	diskioqt.write_count
Bytes read from disk	mmrysize.format(diskioqt.read_bytes)
Bytes written to disk	mmrysize.format(diskioqt.write_bytes)
Time spent reading from disk	timevalu.format(diskioqt.read_time)
Time spent writing to disk	timevalu.format(diskioqt.write_time)
Merged reads	diskioqt.read_merged_count
Merged writes	diskioqt.write_merged_count
Number of disk partitions	len(partlist)
Logical partition device	indx.device
Logical partition usage	{ "free": mmrysize.format(partfree), "used": mmrysize.format(partused), "comp": mmrysize.format(partcomp), "perc": partperc }
Logical partition filesystem information	{ "mtpt": indx.mountpoint, "fsys": indx.fstype }
Physical partition device	indx.device
Physical partition usage	{ "free": mmrysize.format(partfree), "used": mmrysize.format(partused), "comp": mmrysize.format(partcomp), "perc": partperc }
Physical partition filesystem information	{ "mtpt": indx.mountpoint, "fsys": indx.fstype }
CPU utilization percentage	int(cpudperc[indx])
Maximum CPU frequency	freqvalu.format(cpudfreq[indx].max)
Minimum CPU frequency	freqvalu.format(cpudfreq[indx].min)

Current CPU frequency	freqvalu.format(cpudfreq[indx].current)
Time spent in user mode	timevalu.format(cputsecs[indx].user)
Time spent in user mode with low priority	timevalu.format(cputsecs[indx].nice)
Time spent in kernel mode	timevalu.format(cputsecs[indx].system)
Time spent idle	timevalu.format(cputsecs[indx].idle)
Time spent in I/O wait	timevalu.format(cputsecs[indx].iowait)
Time spent handling hardware interrupts	timevalu.format(cputsecs[indx].irq)
Time spent handling software interrupts	timevalu.format(cputsecs[indx].softirq)
Time spent by other operating systems running in a virtualized environment	timevalu.format(cputsecs[indx].steal)
Time spent running a guest operating system	timevalu.format(cputsecs[indx].guest)
Time spent running a niced guest	timevalu.format(cputsecs[indx].guest_nice)
User CPU time percentage	cputperc[indx].user
User CPU time percentage with low priority	cputperc[indx].nice
System CPU time percentage	cputperc[indx].system
Idle CPU time percentage	cputperc[indx].idle
I/O wait CPU time percentage	cputperc[indx].iowait
Hardware interrupt CPU time percentage	cputperc[indx].irq
Software interrupt CPU time percentage	cputperc[indx].softirq
Steal CPU time percentage	cputperc[indx].steal
Guest CPU time percentage	cputperc[indx].guest
Niced guest CPU time percentage	cputperc[indx].guest_nice
CPU name	get_cpu_info().get("brand_raw")
CPU vendor ID	get_cpu_info().get("vendor_id_raw")
CPU frequency	get_cpu_info().get("hz_advertised_friendly")
Number of CPU cores	get_cpu_info().get("count")

CPU architecture (32-bit/64-bit)	<code>get_cpu_info().get("bits")</code>
CPU architecture	<code>get_cpu_info().get("arch")</code>
CPU stepping	<code>get_cpu_info().get("stepping")</code>
CPU model	<code>get_cpu_info().get("model")</code>
CPU family	<code>get_cpu_info().get("family")</code>
CPU feature flags	<code>get_cpu_info().get("flags")</code>
Process ID	<code>str(proc.info["pid"])</code>
Process name	<code>str(proc.info["name"])</code>
Terminal associated with the process	<code>str(proc.info["terminal"])</code>
User running the process	<code>str(proc.info["username"])</code>
Process status (running, sleeping, etc.)	<code>str(proc.info["status"])</code>
CPU usage percentage by the process	<code>%2.1f" % proc.info["cpu_percent"]</code>
Memory usage percentage by the process	<code>"%2.1f" % proc.info["memory_percent"]</code>
Number of threads created by the process	<code>str(proc.info["num_threads"])</code>
Username of the current user	<code>getpass.getuser()</code>
CPU usage percentage	<code>psutil.cpu_percent()</code>
Memory usage percentage	<code>psutil.virtual_memory().percent</code>
Swap memory usage percentage	<code>psutil.swap_memory().percent</code>
Disk usage percentage of the root directory	<code>psutil.disk_usage("/").percent</code>
Percentage of used memory	<code>psutil.virtual_memory().used * 100 / psutil.virtual_memory().total</code>
Percentage of cached memory	<code>psutil.virtual_memory().cached * 100 / psutil.virtual_memory().total</code>
Percentage of free memory	<code>psutil.virtual_memory().free * 100 / psutil.virtual_memory().total</code>
Absolute value of used memory	<code>mmrysize.format(psutil.virtual_memory().used)</code>
Absolute value of cached memory	<code>mmrysize.format(psutil.virtual_memory().cache)</code>

Absolute value of free memory	<code>mmrysize.format(psutil.virtual_memory().free)</code>
Absolute value of total memory	<code>mmrysize.format(psutil.virtual_memory().total)</code>
Absolute value of active memory	<code>mmrysize.format(psutil.virtual_memory().active)</code>
Absolute value of memory buffers	<code>mmrysize.format(psutil.virtual_memory().buffers)</code>
Absolute value of shared memory	<code>mmrysize.format(psutil.virtual_memory().shared)</code>
Absolute value of slab memory	<code>mmrysize.format(psutil.virtual_memory().slab)</code>
Percentage of used swap memory	<code>(psutil.swap_memory().used * 100 / psutil.swap_memory().total) if psutil.swap_memory().total > 0 else 0</code>
Percentage of free swap memory	<code>(psutil.swap_memory().free * 100 / psutil.swap_memory().total) if psutil.swap_memory().total > 0 else 100</code>
Absolute value of used swap memory	<code>mmrysize.format(psutil.swap_memory().used)</code>
Absolute value of free swap memory	<code>mmrysize.format(psutil.swap_memory().free)</code>
Absolute value of total swap memory	<code>mmrysize.format(psutil.swap_memory().total)</code>
Absolute value of swap memory in	<code>mmrysize.format(psutil.swap_memory().sin)</code>
Absolute value of swap memory out	<code>mmrysize.format(psutil.swap_memory().sout)</code>
Percentage of CPU usage	<code>psutil.cpu_percent()</code>
Percentage of free CPU usage	<code>100 - psutil.cpu_percent()</code>
Number of context switches since boot	<code>psutil.cpu_stats().ctx_switches</code>
Number of interrupts serviced since boot	<code>psutil.cpu_stats().interrupts</code>
Number of software interrupts since boot	<code>psutil.cpu_stats().soft_interrupts</code>
Number of system calls since boot	<code>psutil.cpu_stats().syscalls</code>
Process name	<code>procobjc.name()</code>

CPU usage percentage by the process	<code>procobjc.cpu_percent()</code>
Memory usage percentage by the process	<code>procobjc.memory_percent()</code>
CPU number where the process is running	<code>procobjc.cpu_num()</code>
Number of threads created by the process	<code>procobjc.num_threads()</code>
User running the process	<code>procobjc.username()</code>
Terminal associated with the process	<code>procobjc.terminal()</code>
Niceness value of the process	<code>procobjc.nice()</code>
I/O scheduling class of the process	<code>procobjc.ionice().value</code>
Number of context switches experienced by the process	<code>procobjc.num_ctx_switches().voluntary,</code> <code>procobjc.num_ctx_switches().involuntary</code>
Parent process ID	<code>procobjc.ppid()</code>
Process status (running, sleeping, etc.)	<code>procobjc.status().title()</code>
Time when the process was created	<code>procobjc.create_time()</code>
Time when the statistics were acquired	<code>time.time()</code>

Table 3.1: Origin of Statistics

Many more metrics are displayed in the application which has a similar origin. Since a lot of function calls are done repeatedly to acquire these metrics the application gets overhead and a high startup time.

This might be improved by implementing multithreading and also making some changes in the architecture of our application, which is a future scope.

UI Design

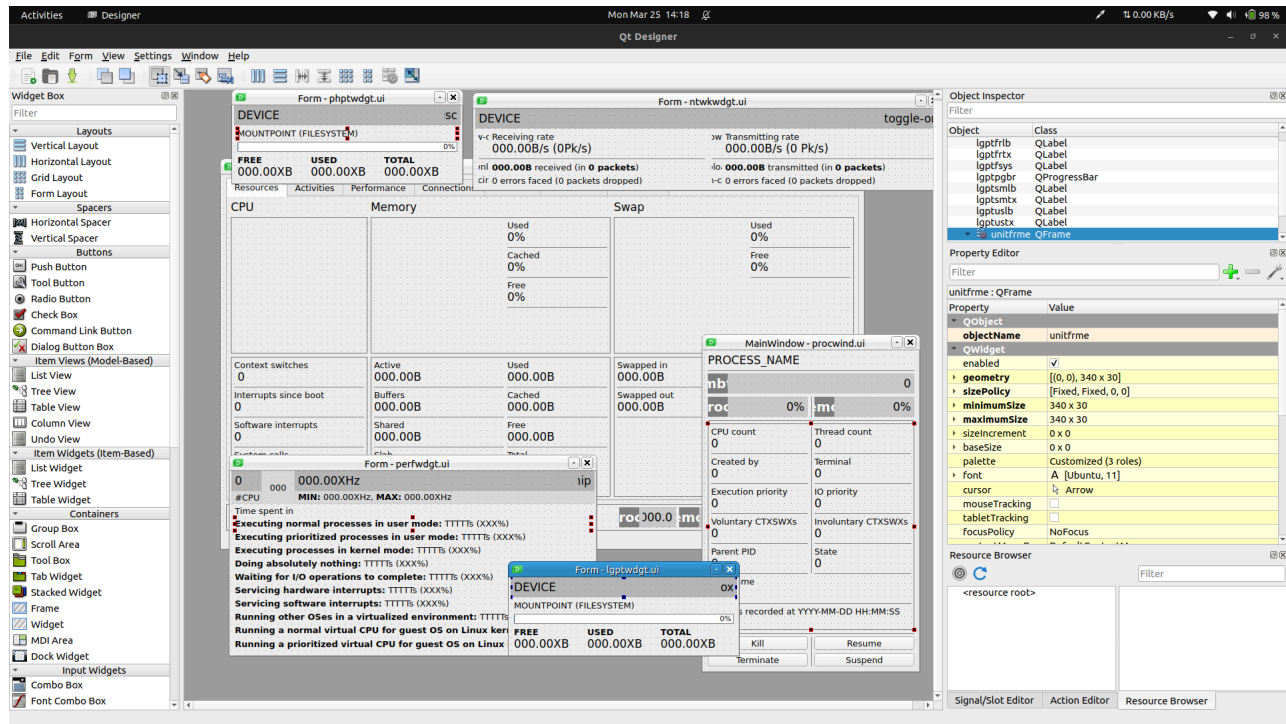


Figure 3.3 : UI Design in Qt Designer

Additionally, we have designed the user interface (UI) for our system resource monitor using Qt Designer, a powerful GUI design tool. Qt Designer facilitates an intuitive design process by offering features such as drag-and-drop functionality and automatic code generation. With Qt Designer, developers can quickly create visually appealing interfaces by arranging widgets and components with ease. The tool generates clean, maintainable code, streamlining the development process and allowing for seamless integration with the backend functionality of the system resource monitor.

Results

Assets Tabscreen

- Displays CPU, Memory, and Swap Memory usage using a radial pie chart.
- Bottom status bar shows CPU Percentage, Memory Percentage, Swap Percentage, and Disk Percentage, along with Linux Kernel Version and owner's name.
- [The number of system calls is always set to 0 in Linux.](#)

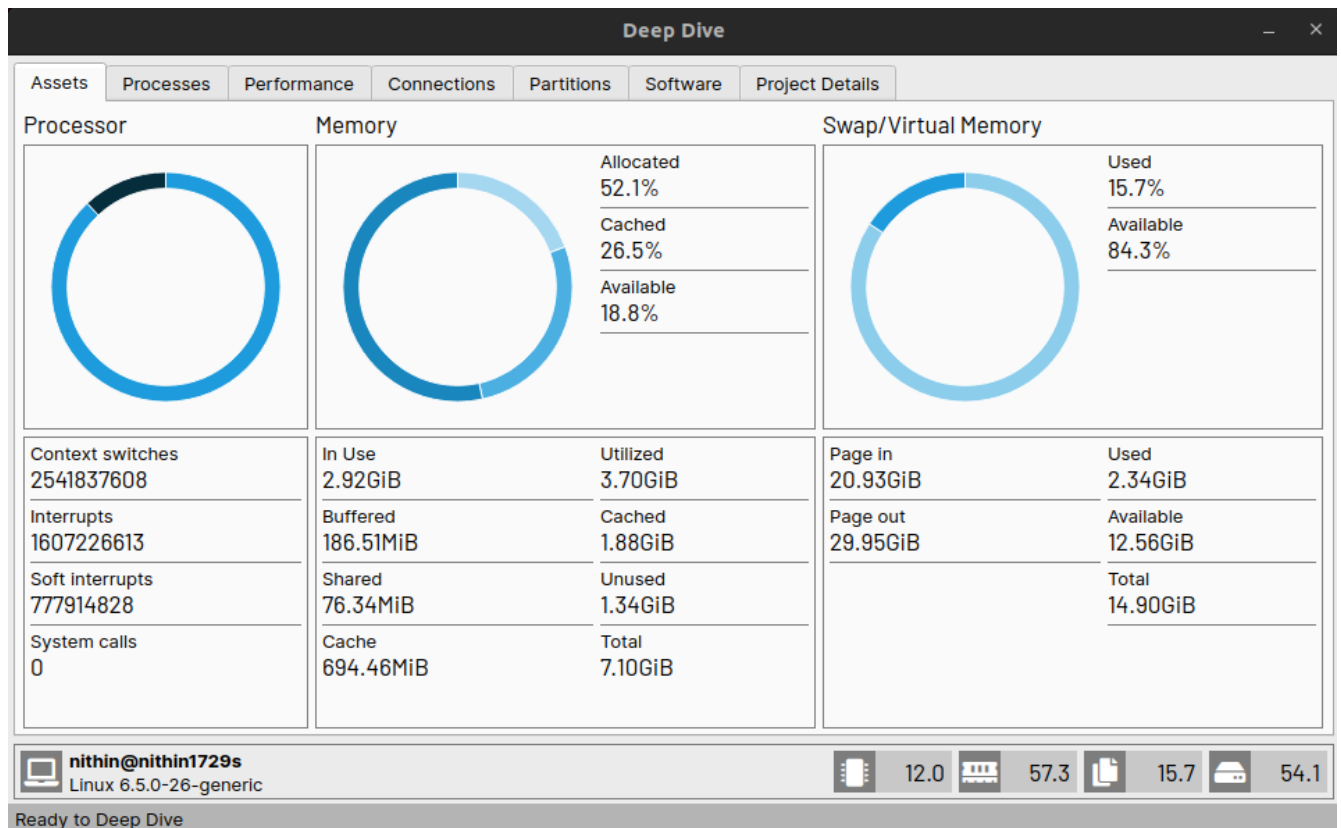


Figure 3.4: Assets Tab Screen

Processes Tabscreen

- Lists processes currently running on the local machine.
- Provides control over terminating, suspending, resuming processes.
- Shows details about the user who created the process, Process ID, Process's Parent ID, CPU % used by the process, etc.

PID	Process Name	Terminal	Creator	State	CPU %	Memory %	Thread #
1	systemd	None	root	sleeping	0.0	0.1	1
2	kthreadd	None	root	sleeping	0.0	0.0	1
3	rcu_gp	None	root	idle	0.0	0.0	1
4	rcu_par_gp	None	root	idle	0.0	0.0	1
5	slub_flushwq	None	root	idle	0.0	0.0	1
6	netns	None	root	idle	0.0	0.0	1
11	mm_percpu_wq	None	root	idle	0.0	0.0	1
12	rcu_tasks_kthread	None	root	idle	0.0	0.0	1
13	rcu_tasks_rude_kthread	None	root	idle	0.0	0.0	1
14	rcu_tasks_trace_kthread	None	root	idle	0.0	0.0	1
15	ksoftirqd/0	None	root	sleeping	0.0	0.0	1
16	rcu_preempt	None	root	idle	0.3	0.0	1
17	migration/0	None	root	sleeping	0.0	0.0	1
18	idle_inject/0	None	root	sleeping	0.0	0.0	1
19	cpuhp/0	None	root	sleeping	0.0	0.0	1
20	cpuhp/2	None	root	sleeping	0.0	0.0	1
21	idle_inject/2	None	root	sleeping	0.0	0.0	1
22	migration/2	None	root	sleeping	0.0	0.0	1
23	ksoftirqd/2	None	root	sleeping	0.0	0.0	1

nithin@nithin1729s
Linux 6.5.0-26-generic

5.9 58.4 15.7 54.1

Ready to Deep Dive

Figure 3.5: Processes Tab Screen

PID #74	
cpuhp/9	
#74	
0.0%	0.0%
CPU # 9	Thread # 1
Created by root	Terminal None
Execution priority 0	IO priority 0
Voluntary CTXSWs 232	Involuntary CTXSWs 232
Parent PID 2	Process State Sleeping
Start time 2024-03-24, 21:21:54	
Metrics acquired on 2024-03-28, 10:48:06.	
Kill	Resume
Terminate	Suspend

Figure 3.6: Process Mini tab Screen

Performance Tabscreen

- Provides information regarding various statistics for each core of the processor.
- Displays CPU Name, Vendor, Count, Architecture, Feature flags, etc.
- Feature flags represent specific capabilities or features supported by the CPU.

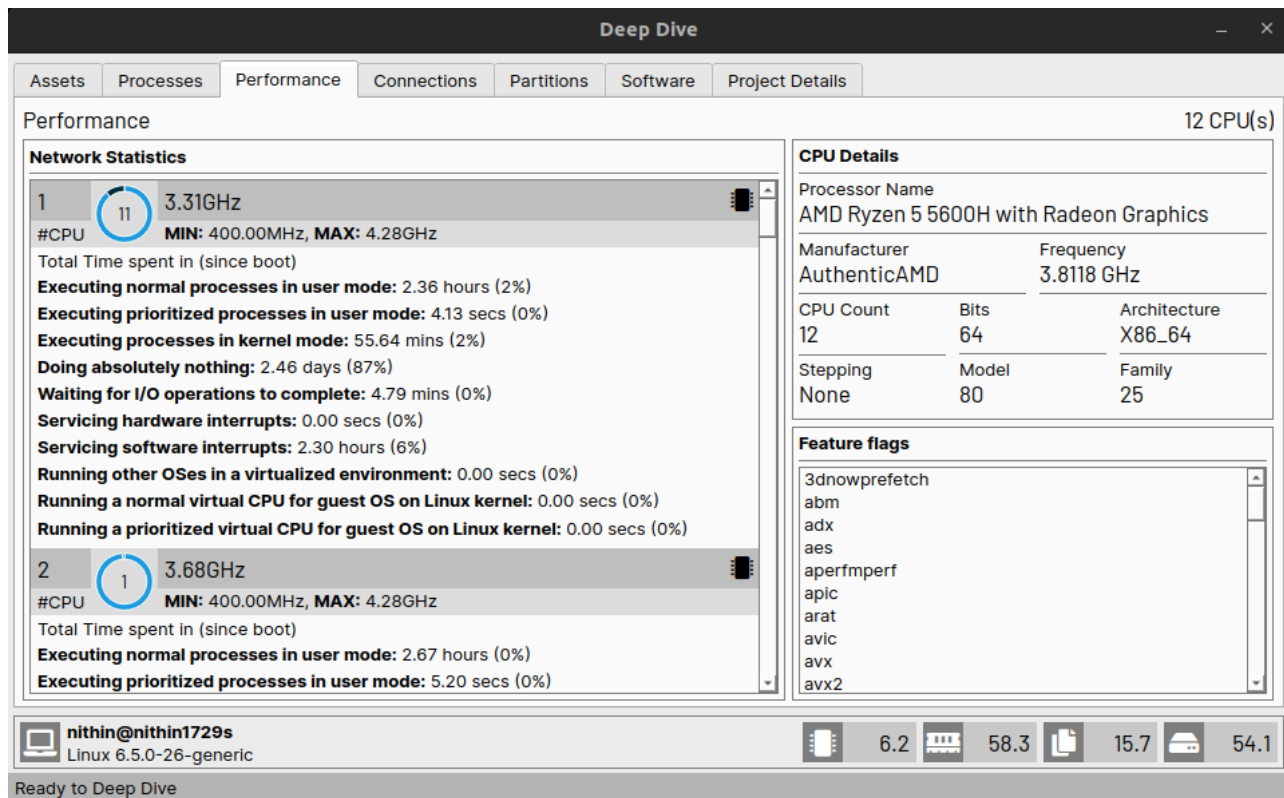


Figure 3.7: Performance Tab Screen

Feature flags, also known as CPU flags or CPU feature flags, are indicators that represent specific capabilities or features supported by a CPU (Central Processing Unit). These flags are used by operating systems, software, and sometimes even by other hardware components to determine the available features and optimize performance accordingly.

Connection Tab Screen

- Displays statistics about internet speed and keeps track of the number of bytes or packets received, and data uploaded or downloaded.
- Includes information about all Network Interface Cards (NIC) present in the system.
- Provides a refresh button to update and view new internet connections.

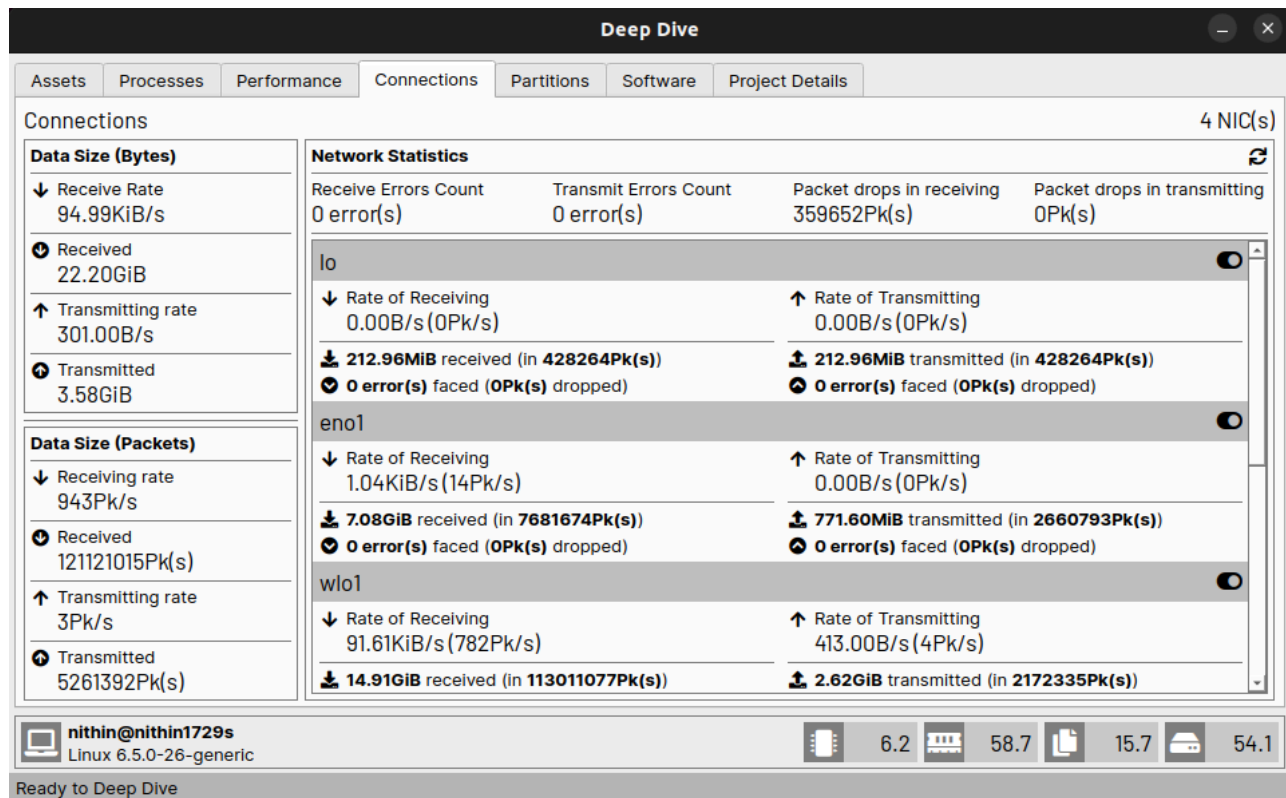


Figure 3.8: Connections Tab Screen

Partition Tabscreen Shows details about the physical and logical partition in the hard disk. It also includes no of read count, write count, merged writes etc

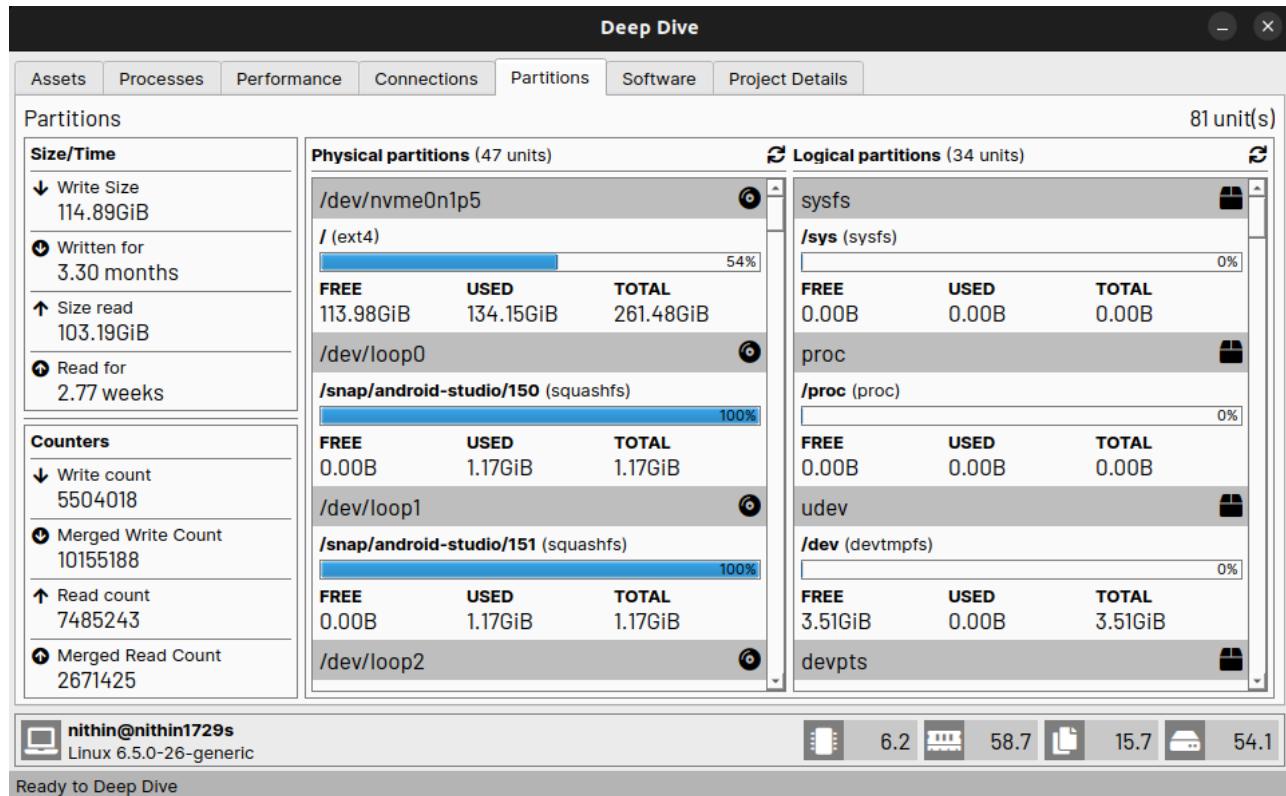


Figure 3.9: Partitions Tab Screen

Information & About Tabscreen Gives information regarding the linux distro and the version of dependencies.

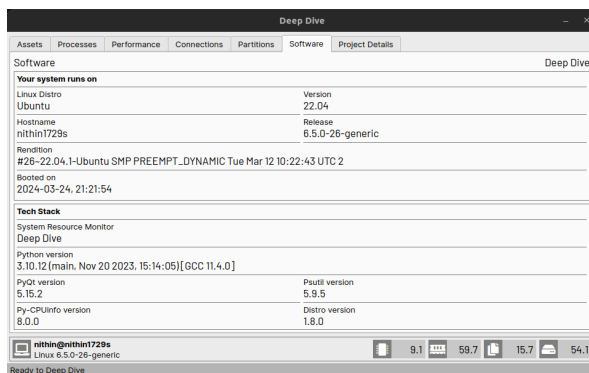


Figure 3.10 : Information Tab Screen

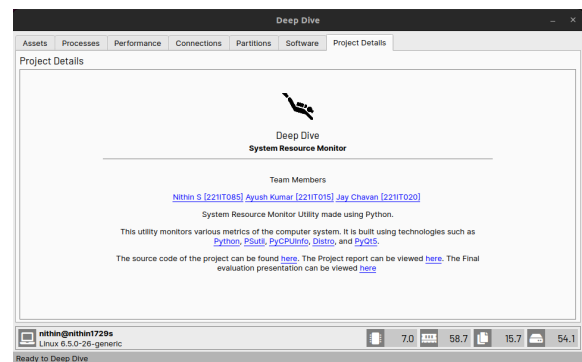


Figure 3.11 : Project Details Tab Screen

CHAPTER 4: CONCLUSION AND FUTURE TRENDS

Conclusion

In conclusion, the system resource monitor we have built represents a significant milestone in the realm of IT infrastructure management and performance optimization. By leveraging advanced monitoring techniques, robust backend functionality, and a user-friendly interface designed using Qt Designer, our system empowers organizations to proactively monitor, analyze, and optimize system resources in real-time.

Future Work

Reduce StartUp Time: Come up with a new architecture which reduces the number of function calls to reduce the start up time of the application. The goal would also be to decrease the overhead of the application.

Enhanced Data Analysis: Introduce advanced statistical analysis and machine learning algorithms to provide deeper insights into system resource usage patterns, enabling predictive analysis for proactive resource management and optimization.

Multi-Platform Support: Extend compatibility to various operating systems beyond the current scope, such as macOS and Linux distributions, to cater to a broader user base and enhance the tool's versatility.

Customization Options: Implement features that allow users to customize the dashboard layout, color schemes, and visualization preferences according to their specific requirements and preferences.

Alerting Mechanisms: Incorporate real-time alerting mechanisms to notify users about critical system resource thresholds being exceeded, enabling timely intervention to prevent performance degradation or system failures.

Historical Data Analysis: Develop functionality to store and analyze historical system resource usage data, facilitating trend analysis, capacity planning, and long-term performance monitoring.

Remote Monitoring and Management: Introduce capabilities for remote monitoring and management, allowing users to monitor system resources across multiple machines from a centralized dashboard and perform management tasks remotely.

Integration with Cloud Platforms: Integrate with popular cloud platforms such as AWS, Azure, and Google Cloud to provide seamless monitoring and management of cloud-based resources, enabling hybrid and multi-cloud environments.

References

PSutil documentation. Available online: <https://psutil.readthedocs.io/en/latest/>

PyCPUInfo documentation. Available online: <https://github.com/workhorsy/py-cpuinfo>

Python os module documentation. Available online: <https://docs.python.org/3/library/os.html>

Distro documentation. Available online: <https://distro.readthedocs.io/en/latest/>