International Journal Of Health, Engineering And Technology (IJHET) *E-ISSN 2829 - 8683*
Volume 5, Number 2, July 2026, *Page. 1541 - 1546*
Email : editorijhess@gmail.com

Analysis Of Cpu And Memory Usage In Monitoring-Based Operating Systems Using Python

Alviyyah Julianti[1*)], Teti Desyani[2)], Tasya Putri Aini[3)], Irvan Maulana[4)], Didin Nugraha[5)], Rayhan Frenalzy[6)]
[1,2,3,4,5)] Informatics Engineering, Pamulang University

*Corresponding Author
Email : alviyyahjulianti@gmail.com , dosen00839@unpam.ac.id , tasaputri69@gmail.com , irvanmaulanaslr344@gmail.com, didinnugraha321@gmail.com, frenalzyrayhan@gmail.com

Abstract

Monitoring computer resource usage, particularly the central processing unit (CPU) and memory, is a crucial aspect in maintaining the performance and stability of an operating system. This study aims to analyze CPU and memory usage in real-time using the Python programming language as the primary tool. The method used involves collecting system resource usage data through Python libraries, such as psutil, followed by data processing and visualization to obtain an overview of usage patterns. The results show that Python is capable of providing accurate and efficient information regarding CPU and memory usage conditions, thus providing a basis for decision-making to optimize system performance. Furthermore, the developed monitoring system is flexible, easy to use, and can be implemented on various operating system platforms. Therefore, this study is expected to contribute to the development of a simple yet effective resource monitoring system. (Monitoring computer resource usage, particularly the central processing unit (CPU) and memory, is a crucial aspect in maintaining the performance and stability of an operating system. This study aims to analyze CPU, and memory usage in real time using the Python programming language as the primary tool. The method used involves collecting system resource usage data through Python libraries, such as psutil, followed by data processing and visualization to obtain an overview of usage patterns. The results show that Python is capable of providing accurate and efficient information regarding CPU and memory usage conditions, thus providing a basis for decision-making to optimize system performance. Furthermore, the developed monitoring system is flexible, easy to use, and can be implemented on various operating system platforms. Thus, this research is expected to contribute to the development of a simple yet effective resource monitoring system.

Keywords: CPU, Memory, Operating System, Python, Monitoring, System Performance.

INTRODUCTION

As information technology continues to develop rapidly, its utilization has become increasingly widespread across various fields, including education, industry, and government. This rapid development has led to a growing dependence on computer systems to support complex and continuous digital activities. In computer implementation, system performance is highly influenced by the utilization of primary resources, namely the CPU and memory. These two components play a crucial role in data processing, program execution management, and maintaining overall system stability to ensure optimal performance (Luthfan Aufar Hindami et al., 2024).

Excessive usage of CPU and memory can have a negative impact on overall computer performance. Such conditions may lead to system performance degradation, including slower response times, reduced processing speed, and even system failure or crashes if not properly managed. In addition, uncontrolled resource usage can disrupt operating system stability and reduce overall computing efficiency. Therefore, monitoring system resource usage is a critical aspect in evaluating real-time computer performance and enabling early detection of potential system issues before they escalate into more serious problems (Z. Afiani & Nurwarsito, 2021).

In current technological development, various methods and tools have been developed to support system monitoring processes. One of the most widely used programming languages for developing monitoring systems is Python. Python is preferred due to its simple syntax, ease of learning, and extensive library support for system resource monitoring and analysis. One commonly used library is psutil, which is capable of retrieving real-time data on CPU usage, memory usage, and running processes accurately and efficiently. By using Python, system monitoring can be implemented

more flexibly and can be deployed across multiple operating system platforms (Syari et al., 2024; Rayendra et al., 2025).

Based on these conditions, this study aims to utilize Python for analyzing and monitoring CPU and memory usage patterns in operating systems. The main focus of this research is to observe fluctuations in system resource usage, particularly under specific conditions such as workload spikes during the initial execution phase or application installation processes. Through this monitoring system, it is expected that more accurate and real-time information regarding system performance can be obtained, which can be used as a basis for decision-making in system performance optimization (Ortiz Villicaña et al., n.d.; P. M et al., 2024).

RESEARCH METHODS

The research method used in this study is observation and experimentation. The observation method was conducted by directly monitoring CPU and memory usage while the system performed various daily operational activities. These activities include application execution, data processing, and background system processes running within the operating system. The purpose of this observation is to obtain a more comprehensive and realistic overview of computer resource usage patterns under different conditions, both when the system is idle and when it is under high workload conditions. This approach allows the researcher to understand how system resources behave in real operational environments.

In addition, the experimental method was applied by varying the system workload. Workload variation was carried out by running multiple applications simultaneously, executing more complex computational processes, and simulating high-demand usage conditions to observe how the system responds to increased activity. During the experiment, changes in CPU and memory usage were systematically observed to determine the system's ability to maintain performance stability under different workload conditions. This step is important to evaluate how efficiently the system manages resources when facing increasing computational demands (Fredella & Rahman, 2025; Apriana et al., 2024).

The monitoring data in this study were collected using a Python-based monitoring program designed to retrieve real-time system information. This program utilizes Python's capability to access system-level data, including CPU usage, memory usage, and active system processes. The data collected are not only momentary snapshots but are continuously recorded over a specific time interval to obtain more accurate and representative patterns of actual system behavior. This continuous monitoring approach ensures that fluctuations in system performance can be captured more effectively.

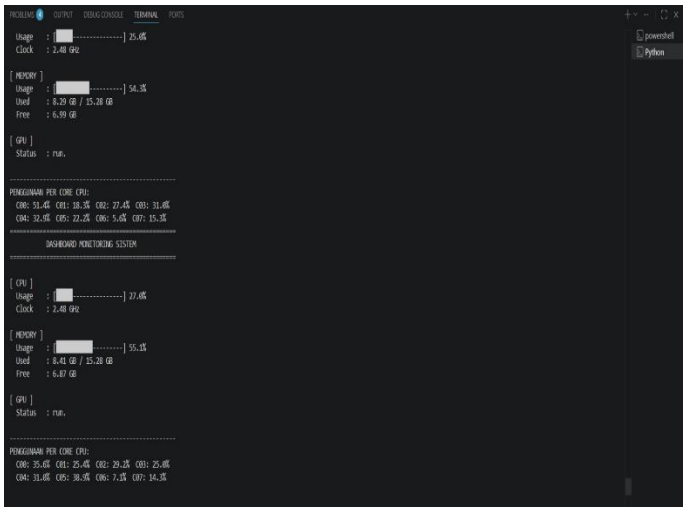
Furthermore, the collected data were analyzed systematically to evaluate overall system performance. The analysis aimed to identify fluctuations in resource usage, understand workload variation patterns, and assess factors that may affect operating system efficiency. This process also helps in detecting critical conditions such as sudden CPU spikes or increased memory consumption, which may potentially impact system stability.

Through this approach, the study does not only focus on data collection but also emphasizes interpretation of results to provide deeper insights into computer system behavior under different operational conditions. This is expected to serve as a strong foundation for developing a more effective and efficient monitoring system capable of providing accurate real-time information for system performance optimization (Ahmed & Jawad, n.d.; Agarwala et al., 2003).

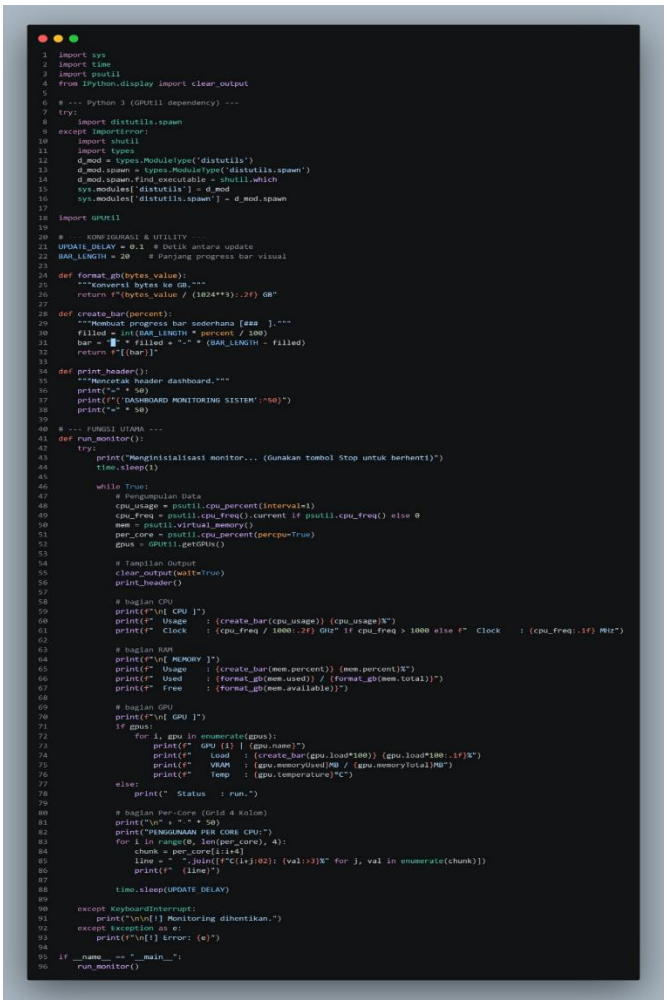
RESULTS AND DISCUSSION

The study conducted monitoring of CPU and memory usage for a duration of 30 seconds using a Python script based on the psutil library. The testing was performed on a laptop with a total RAM specification of 16 GB. Data were collected at regular intervals of a few seconds and visualized in the form of line graphs to better illustrate system resource usage patterns over time.

a) Test 1 (duration ~30 seconds): The average CPU load for User ranged between 11%–33%, while System ranged between 6%–18%. CPU usage showed fluctuations at the beginning, reaching a peak of approximately 33% User load, and then gradually decreased until it stabilized within the range of 12%–19%.
b) Memory footprint remained stable at around 1.8 GB out of a total of 4.0 GB (approximately 45% usage), indicating consistent memory consumption during the monitoring period.
c) Test 2 (duration ~31 seconds): The average CPU load for User ranged between 11%–33%, while System ranged between 7%–15%.
d) A similar pattern was observed, where CPU usage fluctuated at the beginning of the test and then stabilized, with a slight increase observed toward the end of the monitoring period.
e) Memory footprint remained stable within the range of 1.81–1.86 GB, showing only minor variation compared to the first test.
f) The generated graph displays two main lines representing CPU load: the green line (User) indicates processor time used by user-level applications.
g) The red line (System) represents processor time utilized by the operating system kernel.



"Figure 1.1 below shows the output results"



```
import sys
import time
import psutil
from IPython.display import clear_output

# --- Python 3 (GPUtil dependency) ---
try:
    import distutils.spawn
except ImportError:
    import shutil
    import types
    d_mod = types.ModuleType('distutils')
    d_mod.spawn = types.ModuleType('distutils.spawn')
    d_mod.spawn.find_executable = shutil.which
    sys.modules['distutils'] = d_mod
    sys.modules['distutils.spawn'] = d_mod.spawn

import GPUtil

# --- KONFIGURASI & UTILITY ---
UPDATE_DELAY = 0.1  # Detik antara update
BAR_LENGTH = 20     # Panjang progress bar visual

def format_gb(bytes_value):
    """Konversi bytes ke GB."""
    return f"{bytes_value / (1024**3):.2f} GB"

def create_bar(percent):
    """Membuat progress bar sederhana [###  ]."""
    filled = int(BAR_LENGTH * percent / 100)
    bar = '#' * filled + '-' * (BAR_LENGTH - filled)
    return f"[{bar}]"

def print_header():
    """Mencetak header dashboard."""
    print("=" * 50)
    print(f"{'DASHBOARD MONITORING SISTEM':^50}")
    print("=" * 50)

# --- FUNGSI UTAMA ---
def run_monitor():
    try:
        print("Menginisialisasi monitor... (Gunakan tombol Stop untuk berhenti)")
        time.sleep(1)

        while True:
            # Pengumpulan Data
            cpu_usage = psutil.cpu_percent(interval=1)
            cpu_freq = psutil.cpu_freq().current if psutil.cpu_freq() else 0
            mem = psutil.virtual_memory()
            per_core = psutil.cpu_percent(percpu=True)
            gpus = GPUtil.getGPUs()

            # Tampilan Output
            clear_output(wait=True)
            print_header()

            # bagian CPU
            print(f"\n[ CPU ]")
            print(f"  Usage    : {create_bar(cpu_usage)} {cpu_usage}%")
            print(f"  Clock    : {cpu_freq / 1000:.2f} GHz" if cpu_freq > 1000 else f"  Clock    : {cpu_freq:.1f} MHz")

            # bagian RAM
            print(f"\n[ MEMORY ]")
            print(f"  Usage    : {create_bar(mem.percent)} {mem.percent}%")
            print(f"  Used     : {format_gb(mem.used)} / {format_gb(mem.total)}")
            print(f"  Free     : {format_gb(mem.available)}")

            # bagian GPU
            print(f"\n[ GPU ]")
            if gpus:
                for i, gpu in enumerate(gpus):
                    print(f"  GPU {i} | {gpu.name}")
                    print(f"    Load   : {create_bar(gpu.load*100)} {gpu.load*100:.1f}%")
                    print(f"    VRAM   : {gpu.memoryUsed}MB / {gpu.memoryTotal}MB")
                    print(f"    Temp   : {gpu.temperature}°C")
            else:
                print("  Status   : Tidak terdeteksi / Driver tidak ditemukan")

            # bagian Per-Core (Grid 4 Kolom)
            print("\n" + "-" * 50)
            print("PENGGUNAAN PER CORE CPU:")
            for i in range(0, len(per_core), 4):
                chunk = per_core[i:i+4]
                line = "  ".join([f"C{i+j:02}: {val:>3}%" for j, val in enumerate(chunk)])
                print(f"  {line}")

            time.sleep(UPDATE_DELAY)

    except KeyboardInterrupt:
        print("\n\n[!] Monitoring dihentikan.")
    except Exception as e:
        print(f"\n[!] Error: {e}")

if __name__ == "__main__":
    run_monitor()
```

"Figure 1.2 is the input from the program created"

Based on the results of observation and data processing using a Python-based system monitoring instrument, it can be concluded that CPU workload activity has a significant linear correlation with the application execution phase. The fluctuation pattern of resource usage indicates that during application initialization or web page loading, there is a noticeable spike in CPU metrics, representing high computational intensity in code parsing and graphical rendering processes. Conversely, once the application is fully loaded, CPU usage decreases significantly and stabilizes at an idle level, indicating efficient power management by the operating system.

In addition, the monitoring instrument successfully captured the dynamics of main memory (RAM) with high precision, both in terms of used memory and available memory. The data show that RAM allocation changes dynamically depending on the complexity of executed instructions, while available memory remains sufficiently maintained to prevent system bottlenecks. The consistency of data transitions in both processor and memory metrics demonstrates that the developed monitoring method has a high level of sensitivity and accuracy in capturing real-time hardware performance dynamics.

Discussion

Based on the test results, CPU usage on the laptop is not constant but fluctuates over time. These fluctuations are considered normal due to background system processes (such as indexing, updates, or other application activities) that contribute to spikes in both User and System CPU usage. In both tests, the peak CPU usage occurred during the initial phase, which is likely caused by the initialization of the monitoring script itself or other concurrently running processes (Rey et al., 2025).

Memory usage (memory footprint) remained relatively stable throughout the testing period, with only minor variations of approximately 0.05–0.07 GB. This indicates that the developed

monitoring script is lightweight and does not significantly burden system memory. With memory consumption remaining below 2 GB on a 4 GB system, the script is suitable for continuous use as a monitoring tool without degrading overall system performance (Martínez & Rivera, n.d.; Vainio, 2014).

A comparison of the two test scenarios reveals consistent patterns, although slight differences in percentage values were observed. These differences may be attributed to several factors, including:

1. Variations in system conditions during testing (e.g., different background processes),
2. Differences in workload (such as active browser sessions or running applications),
3. Sampling intervals used by the psutil library.

Overall, the implementation of system monitoring using Python and the psutil library successfully produces clear and informative visualizations. This tool helps users understand hardware resource usage patterns in real time and can be utilized for early detection of over-utilization as well as system performance optimization (Cao et al., 2021).

Advantages of this approach include:

1. Simple and easily modifiable code structure,
2. Real-time visualization (live plotting),
3. Cross-platform compatibility (Windows, Linux, macOS).

Observed limitations include: Minor error messages appearing in the console (e.g., “Execution: Unknown word” and “System: Unknown word”), likely due to improperly handled variables or strings, The absence of an automatic export feature for saving graphs as image or PDF files.

CONCLUSIONS

Based on the results of testing and discussion, it can be concluded that the developed Python script is capable of effectively monitoring CPU usage (both User and System) as well as memory footprint in real time. The observed CPU usage shows normal fluctuations, ranging from 11%–33% for User processes and 6%–18% for System processes, while memory usage remains relatively stable at approximately 1.8 GB out of a total of 4.0 GB. The psutil library has proven to be efficient and lightweight for system monitoring purposes, making it suitable for developing simple tools for laptop performance analysis. Furthermore, visualization using Matplotlib provides a clear representation of resource usage patterns, which can assist users in optimizing device performance. Overall, this study demonstrates that hardware monitoring can be performed effectively using open-source tools and the Python programming language without the need for complex or paid software.

For further development, several improvements are recommended. These include adding an automatic data logging feature to CSV files for long-term analysis, implementing threshold-based alerts (such as notifications when CPU usage exceeds 80% or RAM usage exceeds 90%), and expanding monitoring capabilities to include disk usage, network activity, and CPU/GPU temperature. In addition, minor errors in the console output should be resolved to improve readability, and a graphical user interface (GUI), for example using Tkinter or Streamlit, should be developed to make the system more user-friendly.

REFERENCES

Agarwala, S., Poellabauer, C., Kong, J., Schwan, K., & Wolf, M. (2003). *System-level resource monitoring in high-performance computing environments*.

Cao, S., Zeng, Y., Yang, S., & Cao, S. (2021). Research on Python data visualization technology. *Journal of Physics: Conference Series, 1757*(1), 012122. https://doi.org/10.1088/1742-6596/1757/1/012122

Fauzia Fredella, & Rahman, U. (2025). Penerapan virtual memory terhadap kinerja CPU, GPU, dan respons multitasking pada Windows 10. *Mars: Jurnal Teknik Mesin, Industri, Elektro dan Ilmu Komputer*, *3*(5), 168–178. https://doi.org/10.61132/mars.v3i5.1133

Gómez-Luna, J., El Hajj, I., Fernandez, I., Giannoula, C., Oliveira, G. F., & Mutlu, O. (2023). Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware. http://arxiv.org/abs/2110.01709

Karan, N., & Nimay, N. (n.d.). CoreWatch AI-driven CPU/GPU performance analyzer. https://doi.org/10.51583/IJLTEMAS

Martínez, A., & Rivera, C. (n.d.). Enhancing reliability through effective system monitoring.

P. M., A. B. J., & S. E. S. (2024). Real-time web server monitoring system using Python. *Journal of Artificial Intelligence and Capsule Networks*, *6*(3), 332–339. https://doi.org/10.36548/jaicn.2024.3.006

Rey, W. P., Cudilla, E. A. G., & Verdida, R. A. (2025). Performance peaks: Monitoring and optimizing systems for efficiency. *Journal of Advances in Information Technology*, *16*(11), 1595–1603. https://doi.org/10.12720/jait.16.11.1595-1603

Syari, M. A., Ramadani, S., Sembiring, H., & Fitriani, A. (2024). The utilization of Python models in real-time data processing for oil and gas monitoring systems. *Tamika Journal*, *4*(2), 227–231. https://doi.org/10.46880/tamika.Vol4No2(SEMNASTIK)