Federico Cinelli

CS-510: Operating System Principles

**Documentation: Exploring System Resources Using Python**

To successfully run the program, users must first ensure that Python is properly installed on their system. Once Python is available, the next step is to create a virtual environment to keep the project organized and isolated from other dependencies. This can be done by executing the command python -m venv env, which sets up a separate environment folder. After creating the environment, it must be activated. On Windows systems this is achieved with env\Scripts\activate, while Mac and Linux users will use source env/bin/activate. With the environment active, the required dependency, psutil, should be installed by entering pip install psutil in the terminal. After these preparations, the program file should be saved with the correct name main.py and run by typing python main.py. Once executed, the program will display information about CPU usage, CPU count, total memory, and memory utilization in a clean, easy-to-read format.

The design of the program relies on three main functions that work together to retrieve and present important system data. The display\_cpu\_info() function makes use of the psutil library to measure CPU usage over a one-second interval and to determine the total number of logical CPU cores available on the system. This function provides crucial insights because CPU usage percentage reflects how much processing power is currently consumed, while CPU count reveals how many logical units are available for multitasking. The second function, display\_memory\_info(), is responsible for retrieving information about the system’s memory. It calls psutil.virtual\_memory() to collect statistics, then converts the raw byte values into gigabytes for better readability. This function outputs the total amount of memory, the amount currently used, and the percentage of memory consumption. Finally, the main() function serves as the entry point of the program. It prints the title message and then calls both the CPU and memory functions, organizing the data in a way that is clear for the user. By dividing the program into these distinct functions, the structure remains modular, easy to maintain, and simple for others to understand.

The program’s output highlights four main pieces of system information that are essential to understanding performance. First, the CPU usage percentage indicates the proportion of the processor’s resources currently engaged by running processes. A high CPU usage percentage over time may suggest that the system is being pushed to its limits or that specific applications are inefficiently using resources. Second, the CPU count shows how many logical cores are available, which helps determine the system’s ability to handle multitasking and parallel processing. Systems with more cores can better divide workloads, enabling them to perform more tasks simultaneously without slowing down. Third, the program displays the total amount of physical memory available, reported in gigabytes. Finally, the memory usage output shows both the number of gigabytes currently in use and the percentage of total memory being consumed. These combined data points provide a clear snapshot of the computer’s current state and help identify whether resources are underused, balanced, or overextended.

The ability to monitor CPU and memory statistics has practical applications in real-world computing environments. For everyday users, these metrics provide a way to check whether their system is overloaded by background programs or whether performance issues are linked to specific applications. For developers, monitoring CPU and memory usage can reveal bottlenecks in their software and highlight opportunities to optimize performance. For instance, consistently high CPU usage might mean a program needs more efficient algorithms, while high memory consumption could point to memory leaks or poor resource management. System administrators also rely heavily on these metrics when monitoring servers, as they must ensure stable and efficient performance for multiple users. Identifying excessive CPU or memory consumption early can prevent system crashes, reduce downtime, and improve overall reliability. In this way, CPU and memory monitoring is not only helpful for diagnosing problems but also essential for maintaining system health over time.

In addition to monitoring performance, CPU and memory usage also play a central role in how operating systems manage processes and resources. Scheduling algorithms rely on these statistics to distribute CPU time among competing tasks while keeping the system responsive. When CPU usage is high, the scheduler must carefully decide how much processing time each task receives, balancing speed and fairness. Memory usage is equally important, as the operating system must manage both physical and logical memory. Physical memory refers to the actual RAM installed in the machine, while logical memory includes virtual memory, which uses disk space as a temporary extension of RAM. When physical memory runs low, the operating system may resort to paging or swapping, which significantly slows down performance because accessing data on disk is slower than accessing it in RAM. High CPU or memory consumption directly influences these scheduling and allocation decisions, making efficient management critical to ensuring that multiple applications can run smoothly at the same time.

Developing this program came with a few challenges that had to be addressed in order to make the output accurate and user-friendly. One difficulty was dealing with the raw values returned by the psutil library. By default, memory statistics are provided in bytes, which are difficult for most users to interpret. To solve this, the values were converted into gigabytes, which provides a clearer and more familiar scale. Another challenge involved retrieving meaningful CPU usage percentages. The psutil.cpu\_percent() function requires a time interval to properly calculate usage, and without specifying an interval, the results may appear inconsistent or misleading. Ensuring that the program was easy to run in different environments was also a consideration, which meant providing clear setup instructions and verifying that dependencies like psutil were installed. Although these challenges added extra steps to the development process, they ultimately contributed to a more polished and reliable program.