Federico Cinelli

CS-510: Module 2

September 7, 2025

**2-1 Activity: System Utility Processes**

The code retrieves system process information using the psutil library, which provides a convenient interface for accessing details about running processes and system utilization. Specifically, the psutil.process\_iter() function is used to iterate through all currently running processes, requesting process ID, name, status, CPU usage, and memory usage. The program wraps this process in a try-except block to handle any errors that may occur if a process becomes unavailable, is a zombie process, or is restricted due to system permissions. Each process's information is stored as a dictionary and collected in a list for display.

To ensure that all required packages are installed before running the program, a requirements.txt file can be used. This file should include the line psutil, which is the only external dependency needed for this application. To install the dependencies, the user can navigate to the project directory using the command line or terminal and run the following command pip install -r requirements.txt. This will automatically install psutil and any other packages listed in the file, making it easier to set up the environment without manually installing each dependency.

The output of the program is displayed in a tabular format, showing one row per process. For each process, the output includes the Process ID (PID), which is a unique identifier assigned by the system; the Process Name, which indicates the name of the program or service; the Status, which shows whether the processes’ status. The CPU Percent, representing the percentage of the CPU that the process is using, and the Memory Percent, which shows how much of the system’s RAM the process is consuming.

One challenge encountered while creating the program was handling processes that may no longer exist or are restricted due to system permissions. These scenarios can cause the script to raise exceptions if not properly handled. By implementing error handling with try and except blocks, the program can skip these problematic processes without crashing. Another subtle difficulty was ensuring that the CPU and memory usage values are consistently available and correctly formatted, as these values may sometimes be None or fluctuate rapidly depending on system activity.