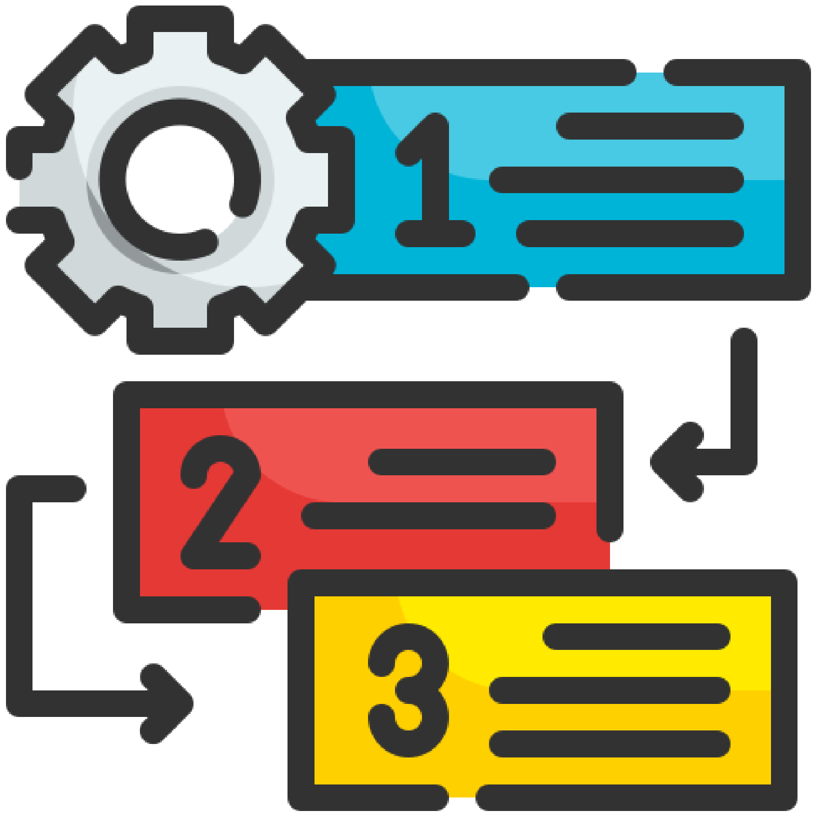

Task Manager Software

Developed by Abdelaziz El-Sheikh

@aelsheikh04

Project Overview

1. Objective

- The goal of this project was to create a cross-platform task manager application to monitor system resources and processes. The application was implemented in both Python and Bash to explore their capabilities and limitations.

2. Approach

- I began with a Python implementation due to its extensive libraries and ease of prototyping. The Python program served as a reference for designing the Bash script.
- The Bash version was developed to replicate the functionality of the Python program while leveraging Bash-specific tools and commands.

3. Features Implemented

- Real-time monitoring of CPU, Memory, Disk, Network, Battery, and GPU usage.
- Display of currently running processes.
- Export of system metrics to an HTML file for analysis.
- Interactive GUI in both implementations (PyQt5 for Python, Zenity for Bash).

Python Task Manager

1. Tools and Libraries Used

- **PyQt5**: For building the graphical user interface (GUI).
- **psutil**: To gather real-time system resource data.
- **matplotlib**: For dynamic graph plotting.
- **GPUtil**: For monitoring GPU usage and status.

2. Key Concepts and Design

- **Object-Oriented Programming (OOP)**: Encapsulation of functionalities in methods and classes (e.g., TaskManager class).
- **Real-Time Monitoring**: Utilized QTimer for periodic updates every second.
- **Multi-Page Design**: Created individual pages for each system metric (CPU, Memory, GPU, etc.) using QStackedWidget.
- **Dynamic Graphing**: Used matplotlib to visually represent trends in CPU and Memory usage.

3. Detailed Features

- **Home Page**:
  - Displayed project title, team members, and navigation buttons.
  - Provided options to save metrics to an HTML file or exit the program.
- **CPU Monitoring**:
  - Displayed real-time CPU usage as a percentage.
  - Included a graph showing CPU usage history over the last 60 seconds.

- **Memory Monitoring**:
    - Provided a real-time memory usage percentage.
    - Showed a graph of memory usage trends.
- **GPU Monitoring**:
    - Displayed GPU details, including memory utilization, temperature, and load.
    - Handled cases where GPU information was unavailable.
- **Processes Page**:
    - Listed running processes with attributes like PID, CPU%, Memory%, and status.
    - Used psutil to retrieve and update process information dynamically.
- **Metrics Export**:
    - Collected system metrics and saved them in an HTML file with a preformatted summary.
    - Ensured error handling during the file-writing process.

2.4 Key Challenges

- Managing multiple GUI elements and ensuring smooth transitions between pages.
- Keeping graphs and metrics updated in real-time without slowing the application.
- Handling exceptions for unavailable hardware (e.g., GPU).

Bash Task Manager

1. Tools and Utilities Used

- **Zenity**: For creating interactive GUI dialogs.
- **feh**: To display images in pop-up windows.
- **awk**, **grep**, and **/proc**: For parsing system data.

2. Key Concepts and Design

- **Shell Functions**: Encapsulated each feature into reusable functions (e.g., cpu_usage, memory_usage).
- **Menu-Driven Interaction**: Used a while loop and case statements to display a main menu and handle user input.
- **Dynamic Data Retrieval**: Accessed live system metrics through /proc and other system utilities.
- **HTML File Creation**: Automated metrics export to an HTML file using Bash scripting.

3. Detailed Features

- **CPU Monitoring**:
  - Calculated CPU usage by reading and parsing /proc/stat.
  - Displayed results in a Zenity dialog with a relevant image.
- **Memory Monitoring**:
  - Read memory stats from /proc/meminfo.
  - Displayed used, free, and total memory in MB.
- **Disk Monitoring**:

  - Used df command to calculate disk usage and free space.
  - Provided formatted output in a dialog box.
- **Network Statistics**:
  - Monitored downloaded and uploaded data using network interface statistics.
  - Displayed the results in MB.
- **Battery Monitoring**:
  - Checked battery status using acpi (if available).
  - Provided an error message if the battery information was unavailable.
- **Processes Page**:
  - Listed top processes sorted by CPU usage.
  - Parsed ps output and displayed it in a formatted Zenity text box.
- **Metrics Export**:
  - Collected system metrics and generated an HTML file. o Dynamically populated the HTML file with CPU, Memory, Disk, and other stats.

4. Key Challenges

- Ensuring compatibility with different Linux distributions.
- Handling edge cases, such as missing GPU or battery data.
- Managing synchronous updates for real-time metrics display.