OPERATING SYSTEM

Project Report

‘GUI Based Task Manager’

Contents

Abstract

This project presents the design and implementation of a **GUI-Based Task Manager System** developed using Python, Tkinter, and Psutil. The system allows users to monitor, manage, and control system processes with features such as process listing, performance monitoring, alerts, automation rules, and historical snapshots. Unlike traditional task managers, this system integrates advanced monitoring, auto-kill rules, and alert logging, providing a comprehensive view of system activity.

1. Introduction

1.1 Overview

Process and resource management is a critical requirement in operating systems. This project implements a Task Manager Application that provides users with a centralized platform to monitor and manage processes efficiently.

1.2 Research Problem

Many task managers either lack advanced monitoring features or do not provide a clear, user-friendly interface for managing processes effectively. The research problem addressed in this project is:

"How can a GUI-based task manager be designed to efficiently monitor, control, and automate process management while maintaining usability and system stability?"

1.3 Research Questions / Hypotheses

- How does real-time process monitoring improve system awareness?
- How do automation rules enhance system stability?
- Can a GUI-based task manager simplify process control for end users?
- Does alert logging improve system diagnostics?

Hypothesis: A task manager integrated with process monitoring, automation, and alert logging provides better system control and usability compared to simple process listing tools.

2. Review of Literature

2.1 Summary of Existing Studies

- Task managers improve system visibility and control.
- Process monitoring enhances diagnostics and troubleshooting.
- Automation rules reduce manual intervention.
- GUI-based systems enhance usability compared to command-line interfaces.

2.2 Positioning the Current Project

Unlike basic task managers, this project provides:

- Real-time process monitoring with alerts
- Automation rules for auto-killing processes
- Historical snapshots of system states
- Alert logging for diagnostics

3. Methodology

3.1 Research Sample

The system environment consists of multiple processes with attributes such as PID, name, CPU usage, memory usage, threads, user, and runtime. Python Tkinter is used for GUI development, and Psutil is used for process and resource monitoring.

3.2 Procedure

- Requirement analysis for process management features
- GUI design using Tkinter
- Implementation of process listing and control (end, suspend, resume, priority)
- Development of monitoring and automation rules
- Integration of alert logging and historical snapshots
- Testing and validation of process operations

3.3 Data Analysis Plan

System data analysis is performed using:

- CPU and memory usage trends

- Process history and snapshots
- Alerts generated during monitoring
- Automation rule triggers

4. System Design and Implementation

4.1 System Architecture

- **Presentation Layer:** Tkinter GUI

- **Application Logic Layer:** Python classes handling process operations, monitoring, and automation
- **Data Layer:** In-memory storage for alerts, history, and snapshots

4.2 Task Model

Each process is represented by attributes:

- PID
- Name
- Status
- CPU usage
- Memory usage
- Threads
- User
- Runtime

4.3 Monitoring Module

Supports:

- Watching specific processes
- Logging alerts when thresholds are exceeded
- Displaying monitored processes in a dedicated tab

4.4 Graphical User Interface

The GUI provides:

- Tabs for processes, performance, system info, startup programs, monitoring, automation, history, and alerts
- Context menus for process control
- Buttons for snapshots, exports, and automation rule management

4.5 Core Functionalities

- End, suspend, resume, and change priority of processes
- Watch processes and monitor resource usage
- Create automation rules for auto-kill
- Take and export snapshots
- Log and export alerts

5. Results

5.1 Functional Validation

All core functionalities were tested and validated successfully. Processes were correctly listed, monitored, and controlled. Automation rules triggered as expected.

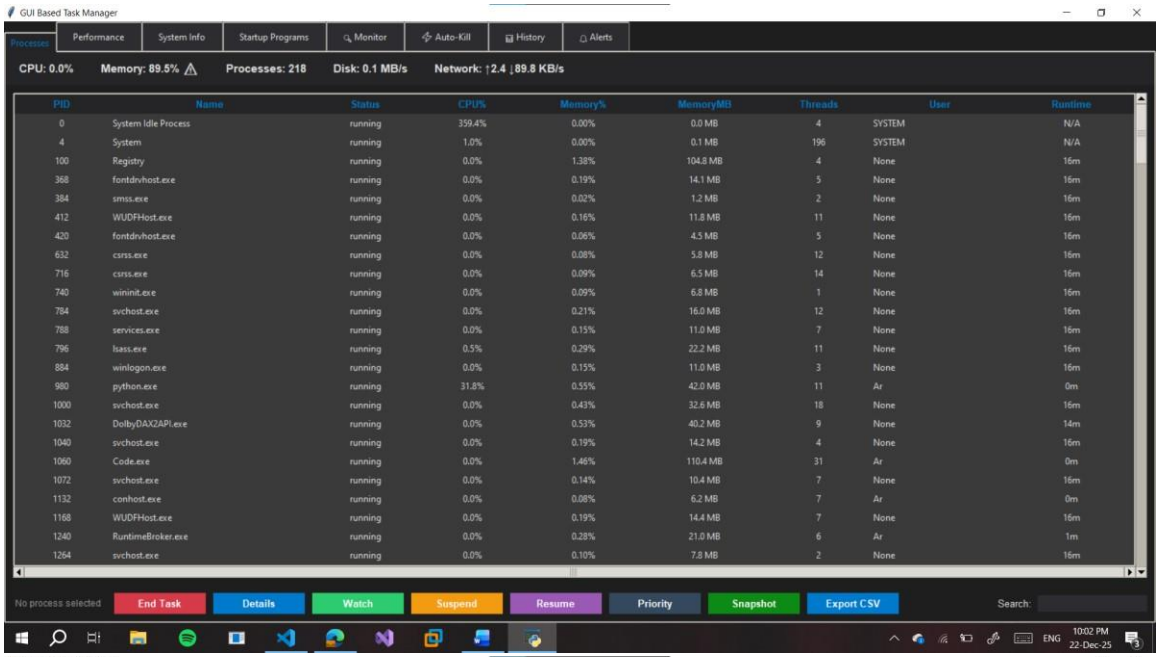
5.2 Performance Observation

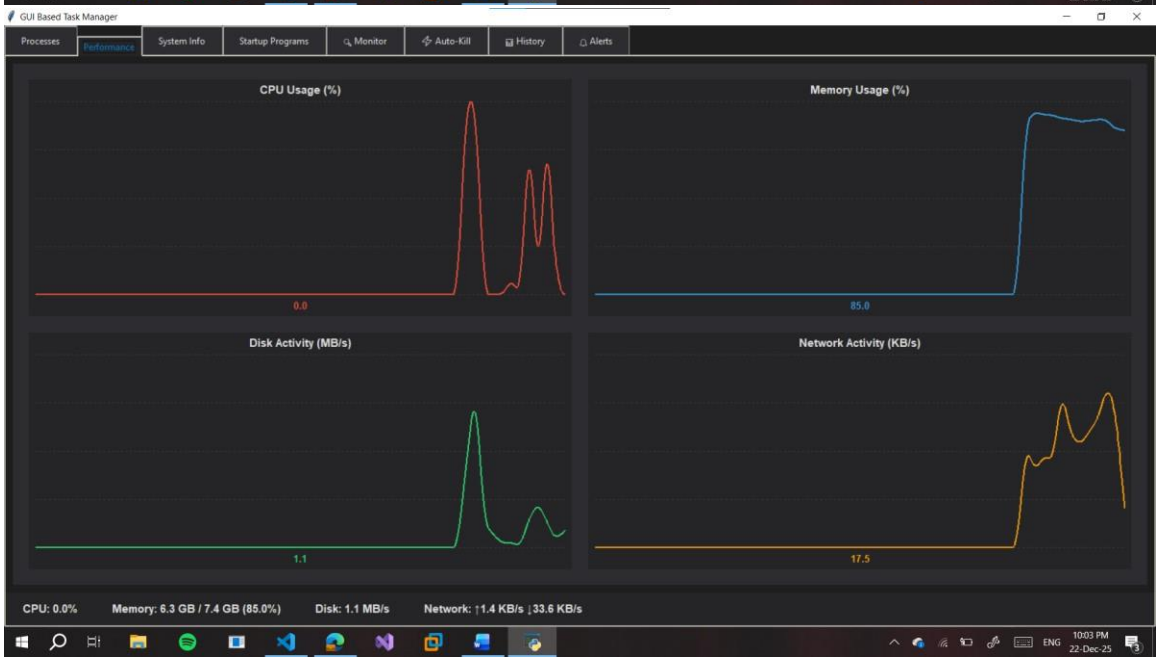
- Real-time updates of CPU, memory, disk, and network usage
- Alerts generated when thresholds exceeded
- Automation rules executed correctly
- Snapshots captured system states

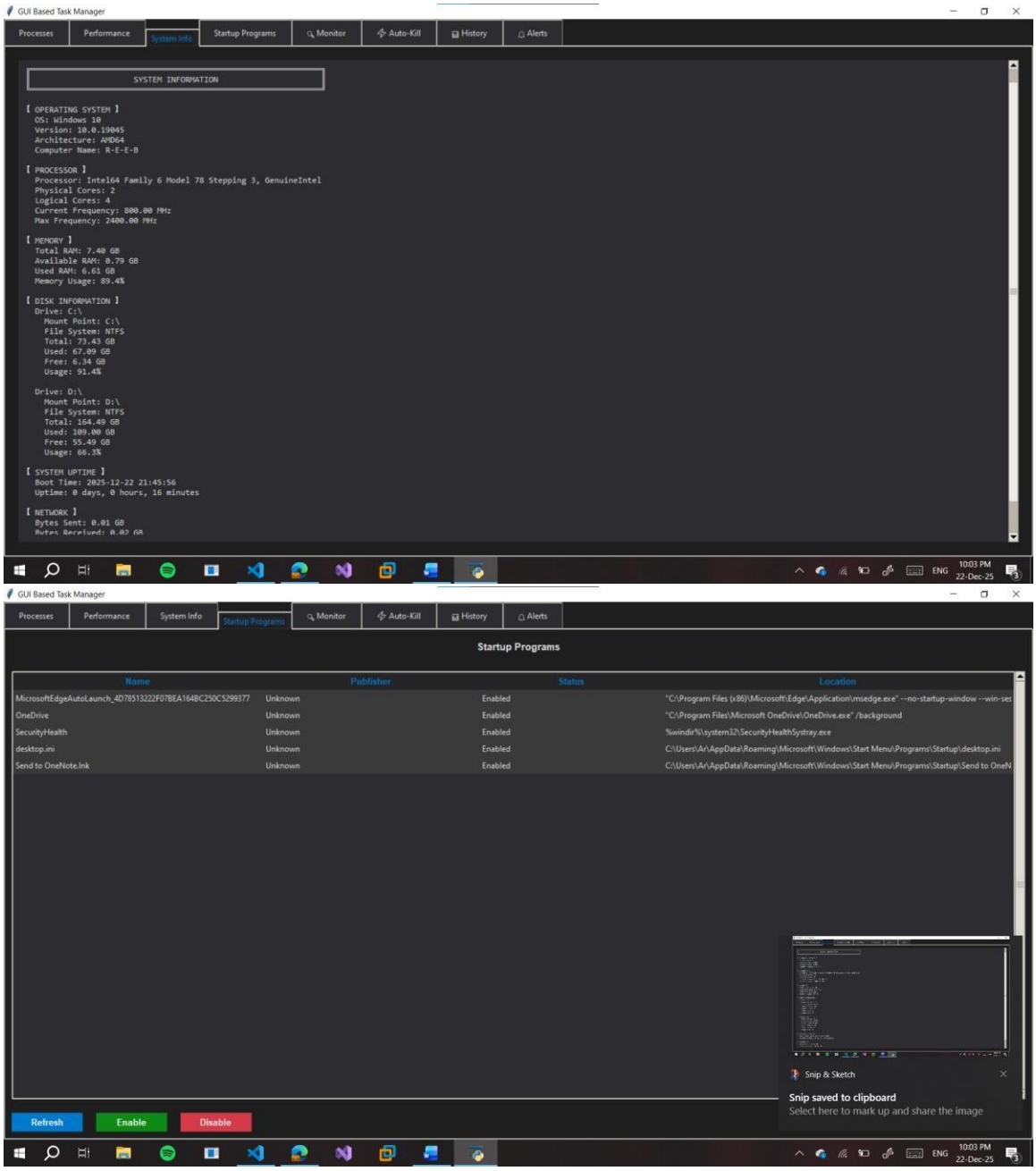
5.3 Usability Evaluation

The GUI effectively simplifies process management operations. Users can monitor and control processes without interacting directly with the operating system, improving accessibility and usability.

6. Screenshots







7. Conclusion

7.1 Summary of Findings

The Task Manager project successfully demonstrates the integration of process monitoring, automation, and alert logging with a desktop GUI application. It provides a secure, efficient, and user-friendly platform for managing system processes.

7.2 Importance

This project strengthens understanding of:

- Process management and operating system concepts
- GUI-based application development
- Real-time monitoring and automation
- System diagnostics and alert logging

7.3 Limitations

- No cloud or multi-device synchronization
- Desktop-only deployment
- Limited automation rule complexity

7.4 Future Recommendations

- Add graphical analytics (charts)
- Support cloud-based or web deployment
- Introduce role-based process monitoring
- Implement advanced alert notifications
