User and System Monitoring Tool Using Shell Scripting

Submitted By

Student Name	Student ID
Munna Biswas	221-15-5261
Mehedi Hasan Saim	221-15-4844
Md. Habibur Rahaman Habib	221-15-5174
Tomalika Sarker	221-15-5341
Lulu Marjan Dina	221-15-5218

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course **CSE324 Operating System Lab in the Computer Science and Engineering Department**

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

December 14, 2024

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Ms. Faiza Feroz**, **Lecturer**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere as lab projects.

Submitted To:

_____

Ms. Faiza Feroz
Lecturer
Department of Computer Science and Engineering
Daffodil International University

Submitted by

<table>
<tr><td colspan="2">_____
Munna Biswas
Student ID:221-15-5261
Dept. of CSE, DIU</td></tr>
<tr><td>_____
Mehedi Hasan Saim
Student ID:221-15-4844
Dept. of CSE, DIU</td><td>_____
Md. Habibur Rahaman Habib
Student ID:221-15-5174
Dept. of CSE, DIU</td></tr>
<tr><td>_____
Tomalika Sarker
Student ID:221-15-5341
Dept. of CSE, DIU</td><td>_____
Lulu Marjan Dina
Student ID:221-15-5218
Dept. of CSE, DIU</td></tr>
</table>

COURSE & PROGRAM OUTCOME

The following course have course outcomes as following:.

Table 1: Course Outcome Statements

CO's	Statements
CO1	**Demonstrate** the ability to implement and work with core operating system concepts such as process management, memory management, and file systems.
CO2	**Develop** Develop and **apply** solutions for synchronization problems (e.g., producer-consumer, readers-writers) using constructs like semaphores, mutexes, and monitors.
CO3	**Simulate** Develop and **evaluate** the performance of CPU scheduling algorithms (e.g., FCFS, SJF, Round Robin) to understand their impact on system efficiency.
CO4	**Develop** and **debug** system-level programs (e.g., shell scripting, inter-process communication) to interact with operating system components effectively.

Table 2: Mapping of CO, PO, Blooms, KP and CEP

CO	PO	Blooms	KP	CEP
CO1	PO1	C1, C2	KP3	EP1,EP3
CO2	PO2	C2	KP3	EP1,EP3
CO3	PO3	C4, A1	KP3	EP1,EP2
CO4	PO3	C3, C6, A3, P3	KP4	EP1,EP3

The mapping justification of this table is provided in section **4.3.1**, **4.3.2** and **4.3.3**.

Table of Contents

Chapter 1

Introduction

This chapter provides an overview of the project, including its background, motivation, objectives, feasibility study, gap analysis, and potential outcomes.

1.1 Introduction

Efficiently managing system resources and user activities is critical but often requires complex tools. This project simplifies these tasks for ease of use and accessibility.

1.2 Motivation

Existing tools are complex or resource-heavy, making monitoring difficult for non-experts. This project offers a streamlined solution, benefiting administrators by saving time and improving productivity.

1.3 Objectives

- Monitor system health (CPU, memory, disk).
- Track user sessions and activity.
- Enable process management.
- Analyze network usage and logs.
- Provide an intuitive, menu-driven interface.

1.4 Feasibility Study

he script leverages Linux utilities like top, ps, and netstat, providing a customizable alternative to resource-heavy tools like Nagios or Zabbix. [**?**].

1.5 Gap Analysis

Existing tools are either complex, resource-intensive. This project fills the gap by offering a lightweight, accessible monitoring solution.

1.6 Project Outcome

The tool provides real-time monitoring with minimal overhead, aids in understanding Linux systems, and is a practical resource for small-scale use or educational purposes.

Chapter 2

Proposed Methodology/Architecture

This chapter details the methodology and design architecture of the "User and System Monitoring Tool Using Shell Scripting" project, covering requirements, system design, UI design, and project planning.

2.1 Requirement Analysis & Design Specification

2.1.1 Overview

The system requires a Linux environment with basic shell utilities like uptime, df, free, ps, netstat, and grep. It integrates these utilities into a unified interface to monitor system health, user activities, and logs.

2.1.2 Proposed Methodology/ System Design

The tool uses a modular approach where each functionality is encapsulated within functions to ensure maintainability and scalability. The main script employs a menu-driven interface allowing users to select from system health monitoring, user session tracking, process management, network usage analysis, and log reporting.

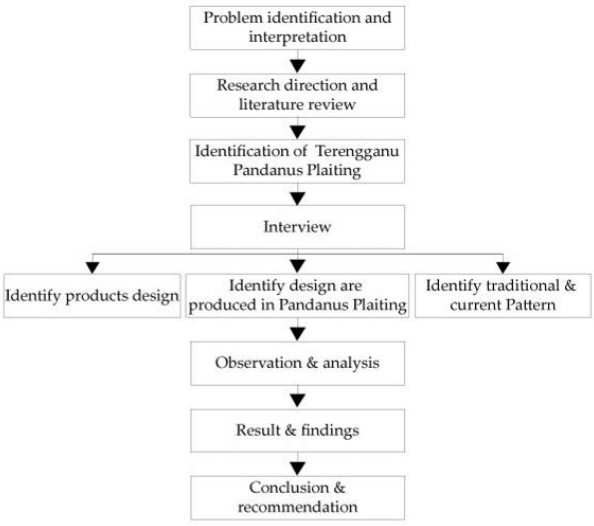

Figure 2.1: This is a sample diagram

2.2 Overall Project Plan

The project follows these phases:

- **Requirement Gathering:** Identifying essential system metrics and user needs.
- **Design Phase:** Creating a modular script structure and drafting a UI design.
- **Implementation:** Writing and integrating shell functions for all monitoring tasks.
- **Testing and Debugging:** Ensuring functionality across various Linux distributions.
- **Deployment:** Making the tool portable and documenting usage instructions.
- **Future Enhancements:** Exploring additional features like email notifications or graphical interfaces.

Chapter 3

Implementation and Results

This chapter outlines the implementation of the "User and System Monitoring Tool Using Shell Scripting," evaluates its performance, and discusses the results achieved.

3.1 Implementation

The project was implemented as a single shell script using Bash, designed to run seamlessly in Linux terminal environments. Key features of the implementation include:

• **Function-based Design:** Each task (e.g., system health, process monitoring) is encapsulated in a dedicated function for modularity.
• **Dynamic Inputs:** Interactive prompts allow users to manage processes or filter log data based on specific criteria.
• **Efficient Resource Usage:** Lightweight Linux commands ensure minimal system overhead.
• **Error Handling:** Built-in checks manage invalid inputs, inaccessible logs, or unsupported commands.

3.1.1 Key Commands and Tools Used

• **System Metrics:** uptime, df, free, and lscpu for CPU, memory, and disk statistics.
• **User Activity:** who and w for session tracking.
• **Process Management:** ps and kill for process monitoring and termination.
• **Network Usage:** netstat and ifconfig for network analysis.
• **Log Analysis:** grep and tail for parsing security and system logs.

3.2 Performance Analysis

The tool was tested on various Linux distributions (Ubuntu, CentOS, Fedora) under different loads to ensure compatibility and efficiency. Performance metrics include:

• **Execution Speed:** All modules provide results within seconds, ensuring real-time monitoring.
• **Resource Usage:** The script has negligible impact on CPU and memory, making it suitable

for low-resource environments.
• **Accuracy:** Outputs from the script were cross-verified against native command results, ensuring reliability.

3.2.1 Observations

• The tool successfully tracked active processes and terminated specified PIDs without errors.
• Network usage monitoring worked effectively, with accurate identification of active connections.
• Log parsing revealed security events promptly, aiding in quick diagnostics.

3.3 Results and Discussion

The tool achieved its primary objectives, providing a unified interface for system monitoring tasks. Results highlight the following advantages:

• **Efficiency:** The modular design and lightweight implementation ensured fast execution.
• **Ease of Use:** The menu-driven interface allowed users to access various functionalities without prior command-line expertise.
• **Scalability:** The script can be extended to include additional features like email alerts or advanced filtering options.

3.3.1 Limitations

• Log parsing depends on consistent log formats, which may vary across Linux distributions.
• Network analysis is limited to basic statistics, lacking deeper packet inspection capabilities.

3.3.2 Discussion

The project demonstrates how simple scripting can deliver effective system monitoring tools. While it is not a replacement for enterprise-level software, its simplicity and adaptability make it valuable for small-scale deployments and educational purposes.

Chapter 4

Engineering Standards and Mapping

This chapter evaluates the project's alignment with engineering standards, its societal and environmental impacts, ethical aspects, and sustainability. It also discusses project management, budget analysis, and how the tool addresses complex engineering problems through a structured framework.

4.1 Impact on Society, Environment and Sustainability

4.1.1 Impact on Life

The tool aids system administrators by simplifying system monitoring and troubleshooting, improving efficiency, and reducing stress during system maintenance.

4.1.2 Impact on Society & Environment

By optimizing resource monitoring and reducing downtime, the tool contributes to minimizing wasteful energy consumption. Its open-source nature promotes knowledge sharing, enhancing societal access to low-cost solutions.

4.1.3 Ethical Aspects

The tool respects user privacy by avoiding unnecessary data collection and limiting system changes to explicitly user-approved actions, ensuring ethical use.

4.1.4 Sustainability Plan

The project is designed to be lightweight, minimizing computational resource demands. By leveraging existing Linux utilities, it aligns with principles of resource reuse and efficiency. Future updates will focus on maintaining compatibility with newer distributions and expanding features without increasing system requirements.

4.2 Project Management and Team Work

4.2.1 Budget Analysis

• **Base Budget:** This assumes minimal costs for development tools (Linux system and open-source software). Approximate cost: 100000 BDT.
• **Alternate Budget:** Adding enhancements like advanced log parsing or integration with monitoring platforms could involve costs for additional tools or APIs. Approximate cost: 20000–50000 BDT.

4.2.2 Revenue Model

The project could adopt an open-source distribution model with optional paid features like email alerts or enterprise support. This hybrid model allows widespread adoption while generating revenue for advanced functionality.

4.2.3 Roles and Responsibilities

The project followed a modular approach, dividing tasks among team members. This division ensured efficiency and timely completion while fostering collaborative problem-solving.

Team Member 1: System Health Monitoring
Objective: Develop a module to track and display essential system health metrics.
Tasks:
• Implement CPU load monitoring using uptime.
• Display memory usage with free -h and disk space usage with df -h.
• Include basic CPU information, such as the model name.
• Deliverables: system health function that outputs CPU load, memory usage, disk space, and CPU model information.

Team Member 2: Active User Session Tracker
Objective: Provide a real-time report on active user sessions.
Tasks:
• Track currently logged-in users and session times using who and w commands.
• Display user session details, including IP addresses for remote logins.
• Deliverables: user sessions function those outputs active user information, including session times and IPs.

Team Member 3: Process Monitoring
Objective: Develop a module for listing and managing system processes.
Tasks:
• Display top CPU-consuming processes.
• Implement a search feature for locating specific processes.

• Include an option to terminate selected processes using the kill command.
• Deliverables: process monitoring function to list processes, search for specific ones, and terminate unwanted processes as needed.

Team Member 4: Network Usage Monitor
Objective: Track and report active network connections and usage.
Tasks:
• List active network connections using netstat -tuln.
• Report network interface statistics, such as received and transmitted packets, using ifconfig.
• Implement threshold alerts for high network usage.
• Deliverables: network usage function that provides detailed network connection information and usage statistics.

Team Member 5: Log Analysis and Reporting
Objective: Analyze system logs for errors, login attempts, and security events.
Tasks:
• Extract and display the last 10 failed login attempts and successful logins from /var/log/auth.log.
• Filter recent system errors from /var/log/syslog or other relevant log files.
• Provide regular log analysis reports for unusual activities.
• Deliverables: log analysis function that analyzes system logs and reports recent errors, login attempts, and security events.

4.2.4 Project Timeline:

Phase	**Description**	**Duration**
Week 1	Requirement Gathering and Tool Setup	1 Week
Week 2	Individual Module Development	1 Week
Week 3	Integration and Testing Development	1 Week
Week 4	Documentation and Presentation Development	1 Week

4.3 Complex Engineering Problem

4.3.1 Mapping of Program Outcome

The problem and its solution align with key Program Outcomes (POs), as shown in Table 4.1.

4.3.2 Complex Problem Solving

The project addresses complex problem-solving categories as mapped in Table 4.2.

Table 4.1: Justification of Program Outcomes

PO's	**Justification**
PO1	Demonstrates problem identification and resolution through system monitoring tools.
PO2	Applies Linux utilities effectively, showcasing technical knowledge and application.
PO3	Encourages innovative scripting and modular design for real-world system management.

Table 4.2: Mapping with complex problem solving.

EP1 Uses foundational Linux knowledge to address real-time monitoring needs.	EP2 Balances ease of use with diverse system requirements.	EP3 Performs detailed log analysis for troubleshooting.	EP4 Familiarity with cross-distribution Linux commands ensures broad usability.	EP5 Incorporates Linux standards like grep, ensuring compliance with known codes.	EP6 Accounts for the role of system admins and stakeholders in tool design.	EP7 Demonstrates interdependence by integrating multiple monitoring features.
√	√	√	√	√	√	√

4.3.3 Engineering Activities

The engineering activities are mapped to key aspects of the project as shown in Table 4.3.

Table 4.3: Mapping with complex engineering activities.

EA1 Range of resources	EA2 Level of Interaction	EA3 Innovation	EA4 Consequences for society and environment	EA5 Familiarity
√	√	√	√	√

Chapter 5

Conclusion

This chapter provides a summary of the project, highlights its limitations, and outlines potential directions for future enhancements.

5.1 Summary

The "User and System Monitoring Tool Using Shell Scripting" successfully addresses the need for an accessible, lightweight, and efficient system monitoring solution. By integrating key Linux utilities into a unified, menu-driven interface, the tool simplifies monitoring tasks such as system health checks, process management, network analysis, and log reporting. The project demonstrates the power of shell scripting in solving real-world problems and showcases modular design principles for ease of development and extensibility.

5.2 Limitation

While the project achieved its primary objectives, a few limitations remain:

• **Limited Cross-Distribution Testing:** Although tested on major Linux distributions, certain commands may behave differently in less common systems.
• **Basic Log Parsing:** The tool relies on simple string matching, which may miss complex patterns or events.
• **Minimal Network Analysis:** The tool does not include advanced features such as packet inspection or traffic visualization.
• **No Automated Alerts:** Real-time notifications (e.g., via email or SMS) for critical events are not implemented.

5.3 Future Work

The project has significant potential for expansion, and future developments may include:

• **Enhanced Log Parsing:** Integration with advanced tools like awk or custom scripts for deeper analysis of system logs.
• **Real-Time Alerts:** Adding automated notifications via email or SMS for critical events or

thresholds.

• **GUI Interface:** Developing a graphical interface to complement the command-line tool, improving accessibility for non-technical users.

Cross-Platform Support: Expanding functionality to include monitoring in macOS or Windows environments.

• **Network Insights:** Incorporating features like bandwidth monitoring and traffic analysis for more comprehensive network usage reports.

• **Plugin System:** Allowing users to add custom modules or scripts to extend functionality further.

References