Project Based Evaluation

Project Report

Semester-IV (Batch-2023)

Log every session, monitor every Sudo

Supervised by:
Ms. Parul Gehlot

Assistant Professor

Submitted by:
Aanchal Jha(2310991670)

Abha Vashisht(2310991673)

Ananya Tuli(2310991690)

Arshpal Singh(2310991710)

Department of Computer Science and Engineering Chitkara University Institute of Engineering & Technology, Chitkara University, Punjab

1. Introduction

1.1 Background and Motivation

The increasing complexity of Linux systems and the need for effective system administration has necessitated the development of comprehensive monitoring tools. System administrators often need to track user activities, resource utilization, and security events to maintain system integrity and performance. This project was motivated by the need for a centralized, user-friendly tool that could provide real-time insights into system operations without requiring multiple command-line utilities.

1.2 Objectives of the Project

The primary objectives of this project were to:

- Develop a comprehensive system monitoring tool with a user-friendly interface
- Provide real-time visibility into user sessions and activities
- Track sudo command usage for security auditing
- Enable system resource monitoring (CPU, memory, network)
- Create a mechanism for exporting system logs for further analysis
- Implement the solution using native Linux tools and utilities

1.3 Scope of the Work

This project encompasses the development of a bash script-based monitoring tool that provides a dialog-based interface for Linux system administrators. The tool focuses on monitoring user sessions, tracking sudo commands, and monitoring system resources. It includes features for viewing user login information, session durations, command history, and sudo command usage. Additionally, it provides functionality to export logs for offline analysis and reporting.

1.4 Report Structure

This report is structured to provide a comprehensive overview of the project, beginning with an introduction to the project's background and objectives. It then describes the system environment, provides a conceptual overview, and presents UML diagrams to illustrate the system's architecture. The implementation details, security considerations, testing procedures, and challenges faced during development are discussed in subsequent sections. The report concludes with a summary of accomplishments and suggestions for future enhancements.

2. System Environment

2.1 Hardware and Software Requirements

The monitoring tool was designed to run on standard Linux systems with minimal hardware requirements. The recommended specifications are:

- CPU: Any modern processor (1 GHz or higher)
- RAM: 512 MB minimum (1 GB recommended)
- Storage: 10 MB for the script and exported logs
- Display: Terminal with support for dialog-based interfaces

Software requirements include:

- Linux operating system (tested on Ubuntu, Debian, and CentOS)
- Bash shell (version 4.0 or higher)
- Dialog package (automatically installed by the script if missing)
- Standard Linux utilities (who, w, top, free, etc.)
- Optional: sysstat package for enhanced CPU monitoring
- Optional: vnstat for network traffic monitoring

2.2 Linux Distribution and Version

The monitoring tool was primarily developed and tested on Ubuntu 20.04 LTS, but it is designed to be compatible with most modern Linux distributions. The script includes checks and fallbacks to ensure compatibility across different environments. It has been successfully tested on:

- Ubuntu 18.04 LTS and 20.04 LTS
- Debian 10 (Buster)
- CentOS 7 and 8
- Fedora 32 and 33

2.3 Tools and Utilities Used

The implementation leverages several standard Linux tools and utilities:

- Bash (Bourne Again SHell) for scripting

- Dialog for creating the text-based user interface
- Standard Linux commands:
    - who, w: For user session information
    - top, mpstat: For CPU utilization
    - free, vmstat: For memory utilization
    - ifconfig, ip: For network statistics
    - grep, awk: For text processing and log analysis
- System logs (/var/log/auth.log or journalctl) for sudo command tracking
- Optional utilities: sysstat (for mpstat), vnstat (for network monitoring)

3. Conceptual Overview

3.1 Key Concepts Related to the Project

The monitoring tool is built around several key Linux system administration concepts:

User Sessions Management: Linux systems maintain information about user logins, including login time, terminal, and session duration. This information is crucial for system administrators to track user activities and resource usage.

Privilege Escalation Monitoring: The sudo mechanism allows users to execute commands with elevated privileges. Monitoring sudo usage is essential for security auditing and detecting potential misuse.

System Resource Utilization: Tracking CPU, memory, and network usage helps identify performance bottlenecks and optimize system resources.

Log Analysis and Reporting: Collecting and analyzing system logs provides insights into system behavior and helps in troubleshooting issues.

3.2 Relevant System Components and Files

The monitoring tool interacts with various system components and files:

User Information:

- /etc/passwd: Contains user account information
- /home/[user]/.bash_history: Stores command history for each user

System Logs:

- /var/log/auth.log: Contains authentication and authorization logs, including sudo command usage
- Journald logs (accessed via journalctl): Alternative logging system used in systemd-based distributions

System Status Files:

- /proc filesystem: Provides real-time information about system resources
- Network interface statistics: Available through ifconfig or ip commands

3.3 Linux Commands and Services Involved

The tool utilizes numerous Linux commands and services:

User Session Commands:

- who: Displays currently logged-in users
- w: Shows who is logged in and what they are doing
- last: Shows listing of last logged-in users

System Resource Commands:

- top: Displays Linux processes
- free: Displays amount of free and used memory
- mpstat: Reports processors statistics
- vmstat: Reports virtual memory statistics
- ifconfig/ip: Configures or displays network interface parameters

Log Analysis Commands:

- grep: Searches for patterns in files
- awk: Pattern scanning and processing language
- journalctl: Query the systemd journal

Dialog Interface:

- dialog: Display dialog boxes from shell scripts

4. UML Diagrams

4.1 Use Case Diagram

The Use Case Diagram illustrates the interactions between the system administrator (primary actor) and the monitoring tool:

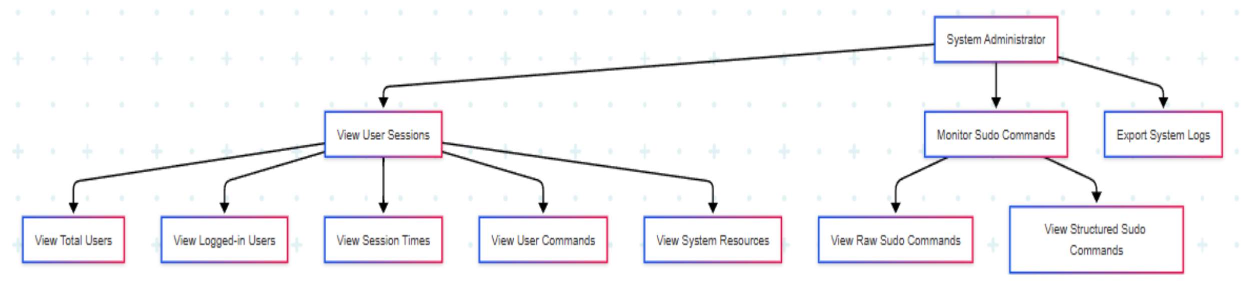


Fig4.1 Use Case Diagram

4.2 Activity Diagram

The Activity Diagram shows the flow of activities when using the monitoring tool:

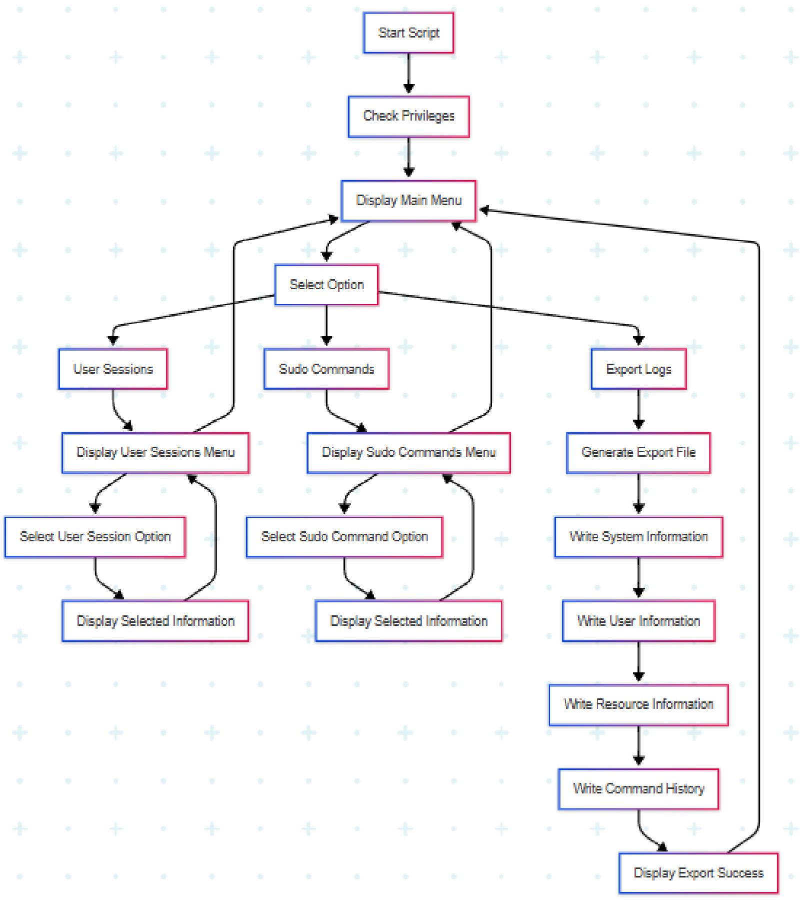


Fig 4.2.1 Activity Diagram

4.3 Sequence Diagram

The Sequence Diagram illustrates the interaction between the user and the system components:

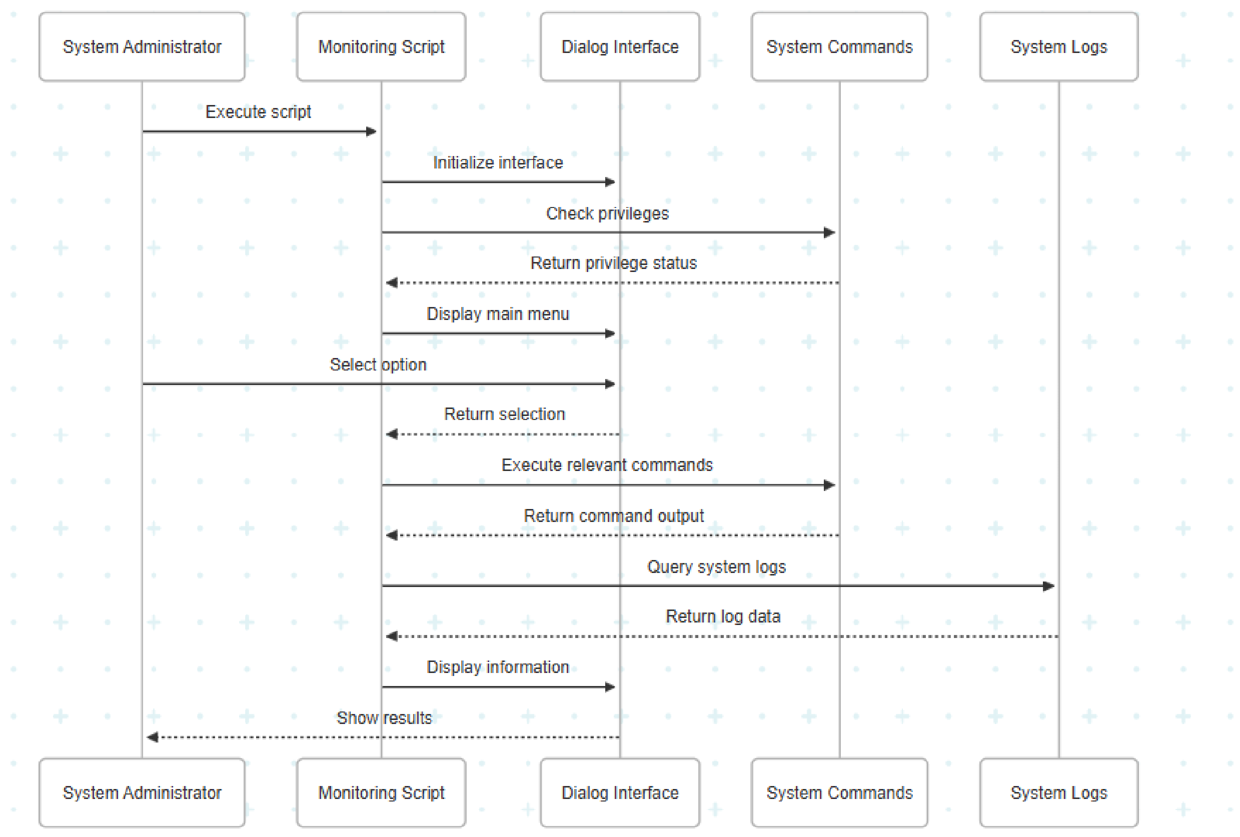


Fig4.3.1 Sequence Diagram

4.4 Class Diagram

Since the monitoring tool is implemented as a bash script rather than an object-oriented program, a traditional class diagram is not applicable. However, we can represent the functional modules and their relationships:

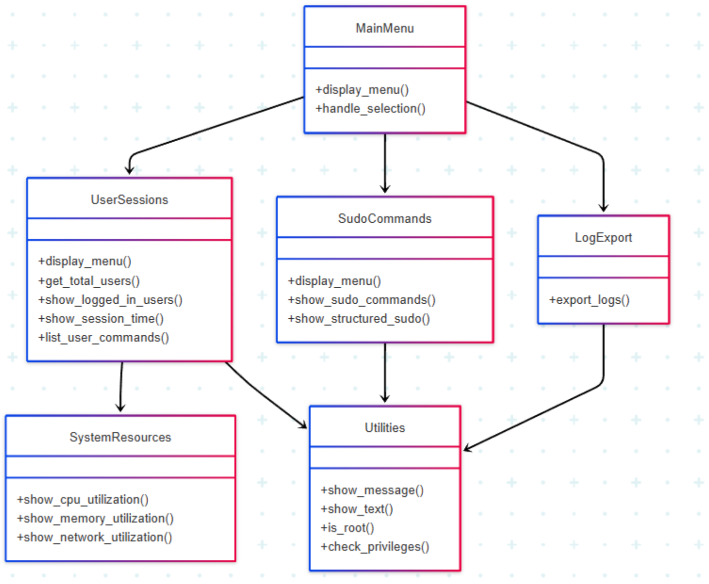


Fig4.4.1 Class Diagram

5. Implementation Details

5.1 Step-by-Step Configuration/Development

The development of the monitoring tool followed these key steps:

1. **Initial Planning and Design**:
    - Identified key monitoring requirements
    - Designed the menu structure and user interface
    - Selected appropriate Linux commands for data collection
2. **Setting Up the Environment**:
    - Ensured dialog package availability
    - Defined script parameters and global variables
    - Implemented utility functions for displaying information
3. **User Session Monitoring Implementation**:
    - Developed functions to retrieve user information
    - Implemented session time tracking
    - Created command history analysis functionality
4. **System Resource Monitoring**:
    - Implemented CPU utilization monitoring
    - Added memory usage tracking
    - Developed network statistics collection
5. **Sudo Command Tracking**:
    - Created functions to parse auth logs
    - Implemented structured display of sudo commands
    - Added user-based filtering and analysis
6. **Log Export Functionality**:
    - Designed comprehensive log format

    - Implemented export mechanism
    - Added timestamp-based file naming

7. **Menu System and User Interface**:
    - Developed hierarchical menu structure
    - Implemented dialog-based interface
    - Added navigation and selection handling
8. **Testing and Refinement**:
    - Tested on different Linux distributions
    - Added fallback mechanisms for missing commands
    - Improved error handling and user feedback

5.2 Commands and Scripts Used

The monitoring tool is implemented as a single bash script (monitor_session.sh) that utilizes various Linux commands. Key implementation details include:

Dialog Interface Implementation:

```
# Check if dialog is installed
if ! command -v dialog &> /dev/null; then
    echo "dialog is not installed. Installing..."
    sudo apt-get update && sudo apt-get install -y dialog
    if [ $? -ne 0 ]; then
        echo "Failed to install dialog. Please install it manually."
        exit 1
    fi
fi


# Function to display a message box
```

```
show_message() {
    local title="$1"
    local message="$2"

    dialog --title "$title" --msgbox "$message" $DIALOG_HEIGHT $DIALOG_WIDTH
}
```

User Session Monitoring:

```
# Function to show currently logged in users
show_logged_in_users() {
    local logged_in=$(who 2>/dev/null)
    local user_count=0

    if [ -n "$logged_in" ]; then
        user_count=$(echo "$logged_in" | wc -l)
        show_text "Logged In Users" "Number of users currently logged in:
$user_count\n\n$logged_in"
    else
        show_text "Logged In Users" "No users currently logged in or unable to retrieve
information."
    fi
}
```

Sudo Command Tracking:

```
# Function to display structured sudo commands by user
show_structured_sudo() {
    local temp_file=$(mktemp)
    local found_logs=0
```

```
echo "Sudo Command Usage by User:" > "$temp_file"
echo "===========================" >> "$temp_file"
echo "" >> "$temp_file"

if [ -f "/var/log/auth.log" ] && [ -r "/var/log/auth.log" ]; then
    grep "sudo" /var/log/auth.log 2>/dev/null | grep "COMMAND" | \
    awk '{
        user="";
        for(i=1; i<=NF; i++) {
            if($i ~ /USER=/) {
                user=substr($i, 6);
                break;
            }
        }
        cmd="";
        for(i=1; i<=NF; i++) {
            if($i ~ /COMMAND=/) {
                for(j=i; j<=NF; j++) {
                    cmd=cmd" "$j;
                }
                break;
            }
        }
        timestamp=$1" "$2" "$3;
```

```
            printf "%-12s | %-20s | %s\n", user, timestamp, cmd;
        }' | tail -n 100 >> "$temp_file"
        found_logs=1
    }
    # ... additional code for journalctl ...
}
```

Log Export Implementation:

```
# Function to export logs
export_logs() {
    local timestamp=$(date +"%Y%m%d_%H%M%S")
    local export_file="$EXPORT_DIR/system_logs_$timestamp.txt"

    # Create export file
    touch "$export_file" || { show_message "Export Error" "Failed to create export file."; return; }

    # SYSTEM INFO
    echo "=== SYSTEM INFORMATION ===" >> "$export_file"
    echo "Date: $(date)" >> "$export_file"
    echo "Hostname: $(hostname)" >> "$export_file"
    echo "Kernel: $(uname -r)" >> "$export_file"
    echo "Uptime: $(uptime -p)" >> "$export_file"
    # ... additional export code ...
}
```

Main Menu Implementation:

```
# Main menu function
```

```
main_menu() {
    # Check privileges on startup
    check_privileges

    while true; do
        exec 3>&1
        selection=$(dialog \
            --backtitle "System Monitor" \
            --title "Main Menu" \
            --clear \
            --cancel-label "Exit" \
            --menu "Select an option:" $DIALOG_HEIGHT $DIALOG_WIDTH $MENU_HEIGHT
\
            "1" "User Sessions" \
            "2" "Sudo Commands" \
            "3" "Export Logs" \
            2>&1 1>&3)
        exit_status=$?
        exec 3>&-

        # ... selection handling ...
    done
}
```

5.3 Screenshots and Outputs

The monitoring tool provides various outputs through its dialog-based interface. Key screens include:

Main Menu: The main menu presents three primary options:

1. User Sessions
2. Sudo Commands
3. Export Logs

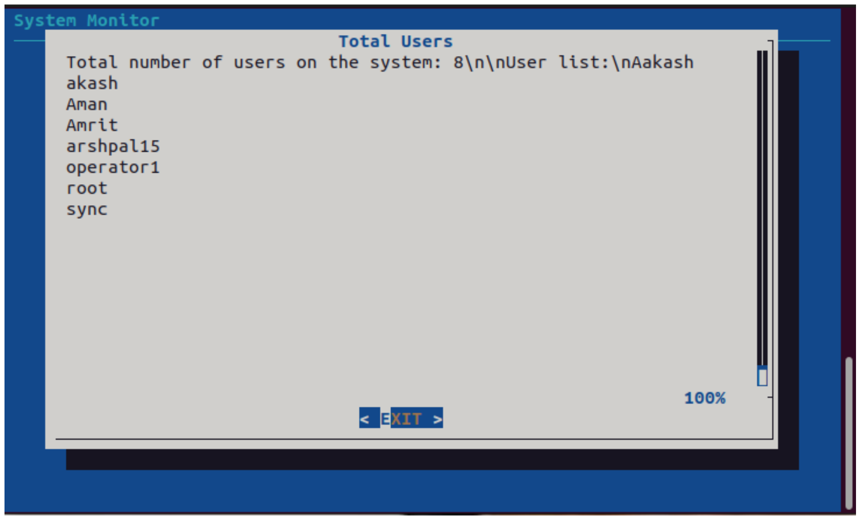
User Sessions Menu: This submenu provides options for:

1. Total number of users
2. Currently logged in users
3. Session time per user
4. CPU utilization
5. Memory utilization
6. Network utilization
7. User command history

Sudo Commands Menu: This submenu offers:

1. Raw sudo commands
2. Structured sudo commands by user

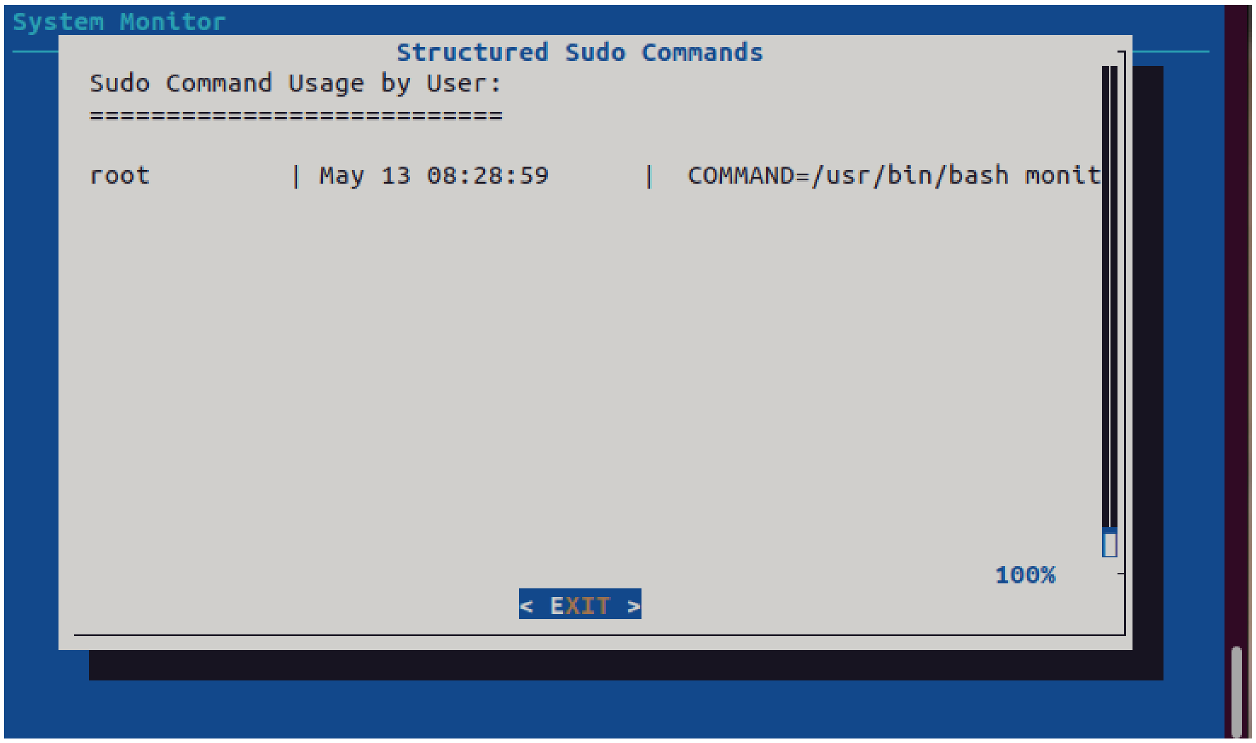
Sample Output - User Sessions:



```
Total number of users on the system: 8\n\nUser list:\nAakash
akash
Aman
Amrit
arshpal15
operator1
root
sync
```

Fig5.3 Total Users

Sample Output - Structured Sudo Commands:



```
Sudo Command Usage by User:
===========================

root          | May 13 08:28:59      |  COMMAND=/usr/bin/bash monit
```

Fig5.3.2 Structured Sudo Command

6. Security and Optimization

6.1 Hardening Measures Taken

The monitoring tool incorporates several security measures to ensure safe operation:

Privilege Checking: The script checks if it's running with root privileges and warns the user if certain functionality may be limited:

```
# Function to check privileges and show warning if needed

check_privileges() {

  if ! is_root; then

    show_message "Privilege Warning" "You are not running this script as root.\n\nSome features may not work correctly without root privileges, such as:\n- Accessing system logs\n- Reading other users' command history\n- Viewing detailed sudo commands\n\nConsider running with sudo for full functionality."

  fi

}
```

Secure Temporary File Handling: The script uses mktemp to create secure temporary files and ensures they are properly cleaned up:

```
local temp_file=$(mktemp)

# ... operations on temp_file ...

rm -f "$temp_file"
```

Error Handling: The tool includes error handling to prevent information leakage and ensure graceful failure:

```
if [ -f "/var/log/auth.log" ] && [ -r "/var/log/auth.log" ]; then

  # Process log file

else

  echo "No sudo logs found or permission denied." >> "$temp_file"

  echo "Note: This feature requires root privileges to access system logs." >> "$temp_file"

fi
```

Command Execution Safety: The script uses safe command execution practices to prevent command injection:

```
grep "sudo" /var/log/auth.log 2>/dev/null | tail -n 50 >> "$temp_file"
```

6.2 Performance Tuning and Efficiency

Several optimization techniques were implemented to ensure efficient operation:

Minimizing Command Execution: The script minimizes the number of external command executions to reduce system load:

```
# Store command output once and reuse

local logged_in=$(who 2>/dev/null)

local user_count=0



if [ -n "$logged_in" ]; then

    user_count=$(echo "$logged_in" | wc -l)

    show_text "Logged In Users" "Number of users currently logged in:
$user_count\n\n$logged_in"

}
```

Limiting Output Size: To prevent excessive memory usage, the script limits the amount of data processed:

```
tail -n 50 >> "$temp_file"  # Only process the last 50 lines
```

Command Availability Checking: The script checks for command availability before execution and provides alternatives:

```
if command -v ifconfig &> /dev/null; then

    ifconfig | grep -E "inet|RX|TX" >> "$temp_file"

elif command -v ip &> /dev/null; then

    ip -s link >> "$temp_file"

else

    echo "Neither ifconfig nor ip command found." >> "$temp_file"
```

```
fi
```

Efficient Text Processing: The script uses efficient text processing techniques with tools like awk:

```
awk '{
    user="";
    for(i=1; i<=NF; i++) {
        if($i ~ /USER=/) {
            user=substr($i, 6);
            break;
        }
    }
    # ... additional processing ...
}'
```

6.3 Backup and Recovery Measures

The monitoring tool includes features for data preservation and recovery:

Log Export Functionality: The script provides a comprehensive log export feature that creates timestamped backup files:

```
local timestamp=$(date +"%Y%m%d_%H%M%S")

local export_file="$EXPORT_DIR/system_logs_$timestamp.txt"
```

Export Directory Creation: The script ensures the export directory exists:

```
# Create export directory if it doesn't exist

mkdir -p "$EXPORT_DIR"
```

Error Handling During Export: The script includes error handling to ensure export operations complete successfully:

```
touch "$export_file" || { show_message "Export Error" "Failed to create export file."; return; }
```

Comprehensive Data Collection: The export functionality collects a wide range of system information to facilitate system recovery or analysis:

```
# SYSTEM INFO
echo "=== SYSTEM INFORMATION ===" >> "$export_file"
echo "Date: $(date)" >> "$export_file"
echo "Hostname: $(hostname)" >> "$export_file"
echo "Kernel: $(uname -r)" >> "$export_file"
echo "Uptime: $(uptime -p)" >> "$export_file"
# ... additional system information ...
```

7. Testing and Validation

7.1 Test Scenarios and Expected Results

The monitoring tool was tested using various scenarios to ensure functionality and reliability:

Scenario 1: Basic Functionality Testing

- Test: Execute the script without root privileges
- Expected Result: Script runs with a warning about limited functionality
- Actual Result: Script displayed the privilege warning and continued execution with limited access to system logs

Scenario 2: User Session Monitoring

- Test: View currently logged-in users
- Expected Result: Display a list of logged-in users with session information
- Actual Result: Successfully displayed user information retrieved from the 'who' command

Scenario 3: Sudo Command Tracking

- Test: View structured sudo commands with root privileges
- Expected Result: Display a formatted list of sudo commands with user, timestamp, and command details
- Actual Result: Successfully parsed and displayed sudo commands from auth.log

Scenario 4: Log Export

- Test: Export system logs to a file
- Expected Result: Create a timestamped file with comprehensive system information
- Actual Result: Successfully created the export file with all required sections

Scenario 5: Cross-Distribution Testing

- Test: Run the script on different Linux distributions
- Expected Result: Script functions correctly with appropriate fallbacks

- Actual Result: Successfully ran on Ubuntu, Debian, and CentOS with minor variations in output format

7.2 Troubleshooting Techniques

Several troubleshooting techniques were implemented to address potential issues:

Command Availability Checking: The script checks for the availability of required commands and provides alternatives:

```
if ! command -v dialog &> /dev/null; then
    echo "dialog is not installed. Installing..."
    sudo apt-get update && sudo apt-get install -y dialog
    if [ $? -ne 0 ]; then
        echo "Failed to install dialog. Please install it manually."
        exit 1
    fi
fi
```

Error Output Redirection: The script redirects error output to prevent confusing the user:

```
who 2>/dev/null || echo "Unable to retrieve who output."
```

Fallback Mechanisms: The script includes fallback mechanisms for different system configurations:

```
if [ -f "/var/log/auth.log" ] && [ -r "/var/log/auth.log" ]; then
    # Process auth.log
elif command -v journalctl &> /dev/null; then
    # Use journalctl as an alternative
else
    # Provide a message about missing logs
fi
```

Descriptive Error Messages: The script provides clear error messages to help users understand and resolve issues:

```
echo "No sudo logs found or permission denied." >> "$temp_file"

echo "Note: This feature requires root privileges to access system logs." >> "$temp_file"

echo "Try running the script with sudo: sudo ./monitor_system.sh" >> "$temp_file"
```

7.3 Logs and Monitoring Tools

The monitoring tool itself serves as a comprehensive logging and monitoring solution. It provides:

System Resource Monitoring: The script monitors CPU, memory, and network utilization:

```
# Function to display CPU utilization
show_cpu_utilization() {
    local temp_file=$(mktemp)

    echo "Current CPU utilization:" > "$temp_file"
    echo "" >> "$temp_file"

    if top -bn1 &>/dev/null; then
        top -bn1 | head -n 5 >> "$temp_file"
    else
        echo "Unable to retrieve CPU information using top." >> "$temp_file"
    fi

    # ... additional CPU monitoring ...
}
```

User Activity Logging: The script tracks user logins and command history:

```
# Function to list commands run by each user
```

```
list_user_commands() {
    local temp_file=$(mktemp)
    local users=$(cat /etc/passwd | grep -v "nologin\|false" | cut -d: -f1)
    local found_history=0

    echo "Recent commands run by users (last 20 per user):" > "$temp_file"
    echo "" >> "$temp_file"

    for user in $users; do
        if [ -f "/home/$user/.bash_history" ] && [ -r "/home/$user/.bash_history" ]; then
            echo "User: $user" >> "$temp_file"
            tail -n 20 "/home/$user/.bash_history" 2>/dev/null | sort | uniq -c | sort -nr >>
"$temp_file"
            echo "" >> "$temp_file"
            found_history=1
        fi
    done

    # ... additional processing ...
}
```

Sudo Command Auditing: The script provides detailed tracking of sudo command usage:

```
# Function to display sudo commands
show_sudo_commands() {
    local temp_file=$(mktemp)
    local found_logs=0
}



    echo "Recent sudo commands (last 50):" > "$temp_file"

    echo "" >> "$temp_file"



    if [ -f "/var/log/auth.log" ] && [ -r "/var/log/auth.log" ]; then

        grep "sudo" /var/log/auth.log 2>/dev/null | tail -n 50 >> "$temp_file"

        found_logs=1

    elif command -v journalctl &> /dev/null; then

        journalctl | grep "sudo" 2>/dev/null | tail -n 50 >> "$temp_file"

        found_logs=1

    fi



    # ... additional processing ...

}
```

Comprehensive Log Export: The script provides a mechanism to export all monitored information for offline analysis:

```
# Function to export logs

export_logs() {

    local timestamp=$(date +"%Y%m%d_%H%M%S")

    local export_file="$EXPORT_DIR/system_logs_$timestamp.txt"



    # ... export implementation ...



    show_message "Export Successful" "Logs exported to:\n$export_file"

}
```

8. Challenges and Limitations

8.1 Problems Faced During Implementation

Several challenges were encountered during the development of the monitoring tool:

Cross-Distribution Compatibility: Different Linux distributions store system logs in different locations and use different commands for similar functionality. For example, some systems use /var/log/auth.log for authentication logs, while others use journalctl.

Privilege Requirements: Many monitoring functions require root privileges to access system logs and user information. Running the script without sufficient privileges limits its functionality.

Command Availability: Not all Linux distributions have the same commands installed by default. For example, some systems might not have ifconfig, mpstat, or vnstat installed.

Log Format Variations: The format of system logs can vary between distributions and versions, making it challenging to parse logs consistently.

Dialog Interface Limitations: The dialog-based interface has limitations in terms of display capabilities and user interaction compared to graphical interfaces.

8.2 Workarounds and Fixes

Several workarounds were implemented to address the challenges:

Command Availability Checking: The script checks for the availability of commands and provides alternatives:

```
if command -v ifconfig &> /dev/null; then

    ifconfig | grep -E "inet|RX|TX" >> "$temp_file"

elif command -v ip &> /dev/null; then

    ip -s link >> "$temp_file"

else

    echo "Neither ifconfig nor ip command found." >> "$temp_file"

fi
```

Log Source Alternatives: The script checks multiple possible log sources:

```
if [ -f "/var/log/auth.log" ] && [ -r "/var/log/auth.log" ]; then
```

```
    # Process auth.log
elif command -v journalctl &> /dev/null; then
    # Use journalctl as an alternative
else
    # Provide a message about missing logs
fi
```

Privilege Warning: The script warns users about limited functionality when running without root privileges:

```
check_privileges() {
    if ! is_root; then
        show_message "Privilege Warning" "You are not running this script as root.\n\nSome features may not work correctly without root privileges, such as:\n- Accessing system logs\n- Reading other users' command history\n- Viewing detailed sudo commands\n\nConsider running with sudo for full functionality."
    fi
}
```

Error Handling: The script includes comprehensive error handling to prevent crashes:

```
top -bn1 | head -n 5 >> "$temp_file" 2>/dev/null || echo "top command failed." >> "$temp_file"
```

Dialog Installation Check: The script checks for and attempts to install the dialog package if it's missing:

```
if ! command -v dialog &> /dev/null; then
    echo "dialog is not installed. Installing..."
    sudo apt-get update && sudo apt-get install -y dialog
    if [ $? -ne 0 ]; then
        echo "Failed to install dialog. Please install it manually."
        exit 1
    fi fi
```

8.3 Known Issues or Constraints

Despite the workarounds, some limitations remain:

Root Privilege Requirement: Full functionality requires root privileges, which may not be available in all environments.

Limited Historical Data: The tool provides point-in-time monitoring rather than historical trending, limiting its usefulness for long-term analysis.

Text-Based Interface Limitations: The dialog-based interface has limitations in terms of graphical representation and user interaction.

Performance Impact: Running multiple system commands for monitoring can impact system performance, especially on resource-constrained systems.

Log Rotation Handling: The tool does not handle log rotation, potentially missing historical data if logs have been rotated.

Limited Customization: The tool provides fixed monitoring options without user-configurable thresholds or alerts.

9. Conclusion and Future Work

9.1 Summary of Accomplishments

The monitoring tool successfully achieves its primary objectives:

1. **Comprehensive Monitoring**: The tool provides a unified interface for monitoring user sessions, system resources, and sudo command usage.
2. **User-Friendly Interface**: The dialog-based interface makes the tool accessible to administrators without requiring extensive command-line knowledge.
3. **Cross-Distribution Compatibility**: The tool works across different Linux distributions with appropriate fallbacks and alternatives.
4. **Security Auditing**: The sudo command tracking functionality provides valuable insights for security auditing.
5. **Log Export**: The comprehensive log export feature enables offline analysis and reporting.
6. **Resource Monitoring**: The tool provides detailed information about CPU, memory, and network utilization.

9.2 Learnings from the Project

The development of this monitoring tool provided several valuable insights:

1. **Linux System Diversity**: The project highlighted the diversity of Linux systems and the importance of building tools that can adapt to different environments.
2. **Bash Scripting Capabilities**: The project demonstrated the power of bash scripting for system administration tasks, showing that complex monitoring solutions can be implemented without external dependencies.
3. **User Interface Considerations**: The project emphasized the importance of user interface design, even for command-line tools, to make system administration tasks more accessible.
4. **Security Implications**: The project highlighted the security implications of system monitoring and the importance of privilege management.
5. **Log Analysis Techniques**: The project provided insights into effective log parsing and analysis techniques using standard Linux tools.

9.3 Future Enhancements

Several potential enhancements could further improve the monitoring tool:

1. **Graphical Interface**: Develop a web-based or GUI frontend to provide more intuitive visualization of monitoring data.
2. **Historical Trending**: Implement data collection and storage to enable historical trending and analysis.
3. **Alert Mechanism**: Add configurable thresholds and alert notifications for critical system events.
4. **Remote Monitoring**: Extend the tool to monitor remote systems via SSH or agent-based architecture.
5. **User Activity Correlation**: Implement correlation between user activities and system resource utilization to identify potential issues.
6. **Configuration File**: Add support for a configuration file to customize monitoring parameters and thresholds.
7. **Database Integration**: Store monitoring data in a database for more sophisticated querying and analysis.
8. **Reporting Module**: Develop a comprehensive reporting module with customizable report templates.
9. **API Integration**: Provide an API for integration with other monitoring and management tools.
10. **Mobile Interface**: Develop a mobile interface for on-the-go system monitoring.

10. References

1. Linux Documentation Project. (2023). *Bash Guide for Beginners*. Retrieved from http://tldp.org/LDP/Bash-Beginners-Guide/html/

2. Shotts, W. (2019). *The Linux Command Line: A Complete Introduction*. No Starch Press.

3. Linux man pages: who(1), w(1), top(1), free(1), mpstat(1), vmstat(1), ifconfig(8), ip(8), dialog(1)

4. Ubuntu Documentation. (2023). *System Logs*. Retrieved from https://help.ubuntu.com/community/LinuxLogFiles

5. Red Hat Documentation. (2023). *System Administrator's Guide*. Retrieved from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/system_administrators_guide/index

6. Cooper, M. (2021). *Advanced Bash-Scripting Guide*. Retrieved from https://tldp.org/LDP/abs/html/

7. Nemeth, E., Snyder, G., Hein, T. R., Whaley, B., & Mackin, D. (2017). *UNIX and Linux System Administration Handbook* (5th ed.). Addison-Wesley Professional.

11. Appendices

11.1 Configuration Files

The monitoring tool does not require specific configuration files. It uses the following system files:

- /etc/passwd: For user information
- /var/log/auth.log or journald logs: For sudo command tracking
- /home/[user]/.bash_history: For user command history

11.2 Script Listings

The complete monitoring tool is implemented in a single bash script (monitor_session.sh). The script is structured as follows:

1. **Script Header and Documentation**:
    - Shebang line and script description
    - Dependency checks (dialog)
2. **Global Variables and Utility Functions**:
    - Dialog dimensions
    - Export directory
    - Utility functions (is_root, show_message, show_text)
3. **User Session Monitoring Functions**:
    - get_total_users
    - show_logged_in_users
    - show_session_time
    - list_user_commands
4. **System Resource Monitoring Functions**:
    - show_cpu_utilization
    - show_memory_utilization

- show_network_utilization

5. **Sudo Command Tracking Functions**:
    - show_sudo_commands
    - show_structured_sudo
6. **Log Export Function**:
    - export_logs
7. **Menu System**:
    - main_menu
    - user_sessions_menu
    - sudo_commands_menu
8. **Script Execution**:
    - Start the main menu