SHELL Program for Log Analysis

Final Project Report

Software System Development—Monsoon 2024

Team Project

Guided By:

Prof. Sai Anirudh Karre

Made By:

Team 30

Shubham Dewangan 2024202005

Lakshay Baijal 2024202006

Vanshika Singh 2024202008

Aryan Prajapati 2024202009

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

1. About the Project

1.1 Problem Domain

Log files are critical to understanding system behaviour, user interactions, and security events. However, interpreting raw log files is a challenging and time-consuming task for system administrators and security professionals due to their sheer size and unstructured format. Without an efficient tool, identifying errors, tracking system performance, and detecting security vulnerabilities becomes cumbersome.

The problem we aim to address in our **Log File Analyzer** project is the lack of a streamlined and user-friendly approach to navigate, interpret, and analyse log files. These files, such as syslog, auth.log, and dpkg.log, play a vital role in:

- Monitoring system health by logging critical events and ongoing processes.
- Tracking authentication activity to detect unauthorised or suspicious login attempts.
- Understanding software package changes through details of installations, updates, or removals.

Administrators and security professionals often face challenges such as:

1. Sifting through large volumes of data for specific information.
2. Identifying the severity of errors and prioritising actions accordingly.
3. Maintaining security by detecting threats early and acting swiftly.

Thus, the **Log File Analyzer** seeks to automate and simplify this process, providing actionable insights and ensuring efficient system management and enhanced security.

1.2 Solution Domain

The **Log File Analyzer** offers an effective solution by providing an intuitive interface to parse, analyse, and interpret log files. This is achieved through several key features and functionalities:

- **User-Friendly Interface**
  - The tool centralizes log analysis into a single interface, allowing users to focus on critical issues without grappling with raw log data.

- Keyword-based searches and intuitive categorization enable efficient navigation of log files, helping users prioritize system performance and security concerns.

- **Categorization and Error Severity Levels**
  - Errors are classified into **severe**, **mild**, and **warning** categories, making it easier to assess the criticality of system issues. This helps users prioritize troubleshooting efforts effectively.
- **Log Analysis Techniques**
  - **Regular Expressions (Regex)**: Regex patterns are employed to extract meaningful data from log files, such as timestamps, process names, and user authentication details.
  - This pattern-based analysis highlights trends, detects anomalies, and helps pinpoint the root causes of system issues.

- **Advanced Log File Insights**
  - Beyond basic log parsing, the tool provides:
    - **Memory and Process Analysis**: Highlighting high memory usage, system resource bottlenecks, and active/stopped processes.
    - **Authentication Analysis**: Spotting repeated failed logins or brute-force attack attempts.
    - **Package Manager Tracking**: Offering clear insights into changes in system configuration over time.

- **Support for Custom Log Files**
  - The analyzer extends flexibility by enabling users to upload custom log files, regardless of format. Custom regex patterns can be constructed to handle these logs, making the tool adaptable to unique user needs.

- **Enhanced Automation and Simplification**
    - The project automates the tedious process of manually reviewing logs, helping users quickly identify system errors, performance issues, or security concerns.
    - Reports are presented in a clean, structured format with options to export results for further review or archiving.

1.3 System Domain

The Log File Analyzer is a versatile, cross-platform tool designed to support a wide range of operating systems, enabling comprehensive log analysis across various environments. It adapts to the unique log structures of different operating systems, including Linux (parsing logs like syslog, dpkg.log, and auth.log), macOS (handling system logs such as system.log and asl.log), Windows (analyzing Event Logs and custom log formats), and Solaris (processing syslog and other platform-specific files). In addition to predefined log types, the tool supports user-provided custom log files from any OS, dynamically adapting to different file formats through customizable regex patterns for tailored analysis. Furthermore, it allows seamless script execution to process logs and display results clearly. Examples include memory and process analysis scripts for Linux, security analysis for authentication logs in macOS, and custom log parsers for Windows Event Logs.

2. Subsystems and dependencies

2.1 Shell

The core of our Log File Analyzer project is built around the shell environment, which serves as the foundation for executing commands, running scripts, and displaying outputs. The project utilizes Bash shell scripting to perform log analysis, ensuring seamless interaction with the operating system.

Key aspects of shell usage in the project include:

- **Command Execution**: The shell enables the execution of various command-line utilities and scripts for parsing, filtering, and analyzing log files.
- **Output Display**: Analysis results are presented directly on the shell, making it a single-point interface for both execution and visualization.
- **Cross-Platform Compatibility**: By leveraging shell scripting, the project achieves compatibility across Unix-like systems, including Linux, macOS, and Solaris.

The reliance on shell scripting ensures efficiency, automation, and adaptability, empowering users to perform real-time and comprehensive log analysis through an accessible and robust interface.

2.2 Command-Line Tools

The project extensively employs **Bash shell utilities** to process and analyze log files effectively. Below is a brief overview of the tools and their specific roles:

grep

- **Command**: grep 'pattern' logfile
- **Description**: Searches for specific patterns in log files, such as error messages, login attempts, or warnings. Efficiently highlights relevant lines from large datasets.

awk

- **Command**: awk '{print $1, $3}' logfile

- **Description**: Extracts and formats specific columns from log entries, such as timestamps or error codes, enabling detailed reporting and data organization.

sed

- **Command**: sed 's/old/new/g' logfile
- **Description**: Performs stream editing to clean or modify log entries. It can replace text, delete lines, or insert new content, making logs easier to analyze.

sort and uniq

- **Commands**:
  - sort logfile | uniq -c
  - sort logfile | uniq
- **Description**: Summarizes and aggregates log data by organizing entries and eliminating duplicates. Useful for counting occurrences of specific events or identifying frequently accessed resources.

cut

- **Command**: cut -d ' ' -f 1,3 logfile
- **Description**: Extracts targeted fields, such as IP addresses or specific error codes, using delimiters for focused analysis.

tail

- **Command**: tail -f logfile
- **Description**: Monitors log files in real time by displaying the last few lines and updating as new entries are added. Ideal for identifying immediate issues.

By combining these tools, the project achieves a comprehensive and efficient log analysis workflow, handling diverse tasks ranging from simple filtering to complex data manipulation.

2.3 Visualization

To enhance the presentation and interpretation of log analysis results, the project integrates **GNUplot**, a powerful and flexible graphing utility. GNUplot is utilized to visually represent the processed log data, helping users gain actionable insights through clear and informative graphs. This integration brings significant value to the Log File Analyzer, making it easier to understand complex data and identify trends, issues, and patterns in the logs.
Key features of **GNUplot** in this project include:

- **Dynamic Graph Generation**: As the log analysis processes various data, such as error trends, memory usage, or resource utilization, the results are automatically plotted into graphs. These graphs help users quickly spot patterns, trends, and potential anomalies, simplifying the process of analyzing large amounts of log data.
- **Ease of Use**: The graphs are generated automatically, saving users time and effort. They can be easily customized to suit different needs, whether that means creating line charts to show trends over time, bar charts for comparing categories, or scatter plots for visualizing data distributions. This flexibility ensures that the right type of graph can be chosen for each situation.
- **Export Options**: Once the graphs are generated, they can be exported in multiple formats, such as PNG, SVG, or PDF. This feature is particularly useful for documentation, reports, or sharing the visualized data with colleagues or clients. Users can keep a visual record of the log analysis results or share them as part of their findings.

By incorporating **GNUplot**, the Log File Analyzer turns raw, often hard-to-interpret log data into intuitive and visually engaging insights. With the help of graphs, users can easily detect issues, observe trends over time, and make informed decisions based on the visualized data. This makes the tool not only more effective but also easier to use, as it simplifies the process of turning complex data into clear, actionable information.

3. Analysis and Design

3.1 Data flow diagram:

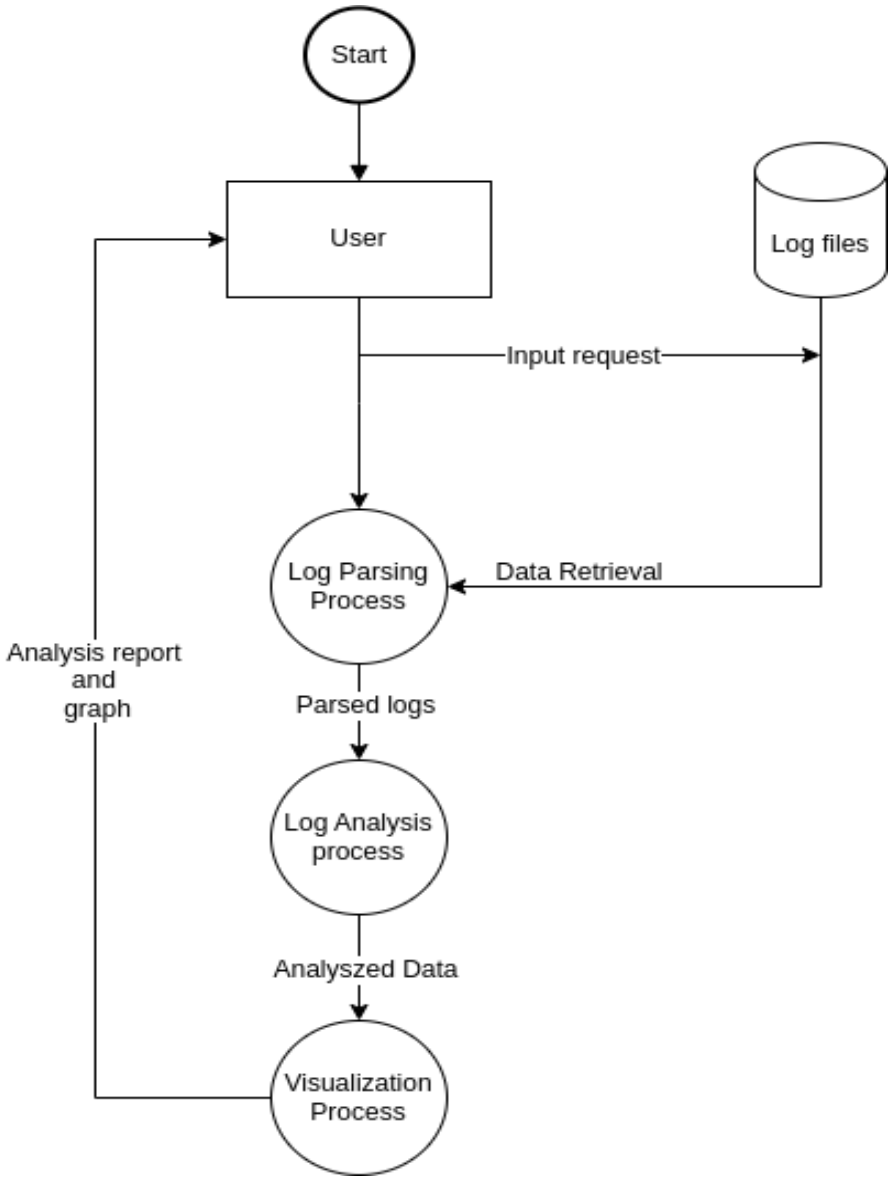


Fig 3.1 Data Flow Diagram for Log Analysis

The Data Flow Diagram (DFD) illustrates how data moves through the log analysis system, highlighting the interaction between the user, processes, and data storage. The steps taken are defined as following :

1. **User Interaction**:
   The process begins when the user interacts with the system, either by selecting log files or initiating the log analysis through the Command-Line Interface (CLI). This provides the user with a straightforward way to start the analysis.

2. **Log Storage**:
   Once the log files are selected, they are fetched from the log storage, such as the common log directories (e.g., /var/log/syslog), and passed on to the next stage, where they will be processed for further analysis.

3. **Log Parsing Process**:
   In this step, the system parses the raw log data, extracting essential information such as error messages, warnings, timestamps, and application-specific events. This ensures that only the most relevant data is retained for further analysis.

4. **Log Analysis Process :**
   The parsed log data is then sent to the log analysis module, where it is carefully examined to identify trends, anomalies, and performance issues. This step is crucial for detecting underlying problems or patterns in the system's behavior.

5. **Visualization Process**:
   After the analysis is complete, the results are passed to the visualization module. Tools like Spark are used to generate graphical representations, such as error frequency trends or application performance metrics. These visual insights make it easier to understand the data at a glance.

6. **Output Delivery**:
   Finally, the analyzed and visualized insights, including text-based summaries and graphical trends, are delivered back to the user through the CLI. The user can then review the results, make informed decisions, and take necessary actions based on the analysis.

3.2 Sequence Diagram:

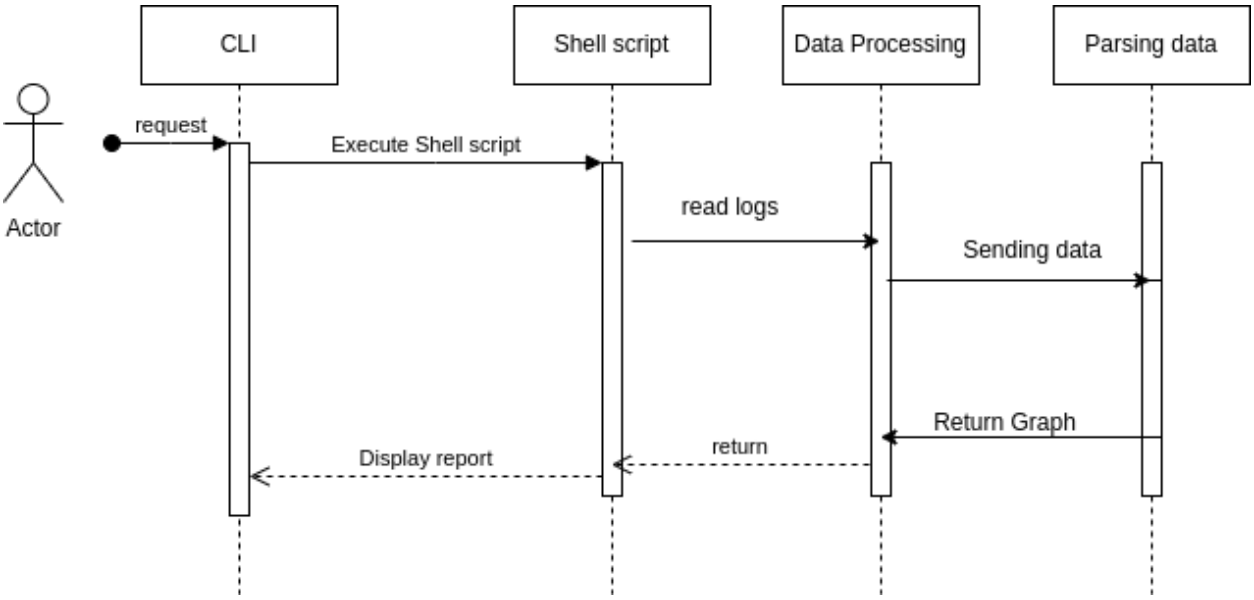


Fig 3.2 Sequence Diagram for Log Analysis

The sequence diagram outlines the step-by-step interaction between the components of the log analysis system, focusing on how the system processes user input and delivers insights. The details of which are described as follow :

1. **User Interaction**:
   The process begins when the user requests log analysis through the Command-Line Interface (CLI). The user specifies the log files or the type of analysis they wish to perform, such as identifying errors, trends, or performance issues. The CLI serves as the entry point, receiving input from the user to trigger the analysis process.

2. **Log Retrieval**:
   Once the user's request is received, the CLI forwards it to the log storage system. The system retrieves the specified log files, typically from directories like /var/log/syslog or other application-specific locations, ensuring that the correct data is available for further processing.

3. **Log Parsing**:

   The retrieved log files are then sent to the log parsing module. Here, the raw log data is processed to extract relevant information such as timestamps, errors, warnings, and other key events. This step ensures that only important details are retained, allowing for efficient analysis in the next phase.

4. **Data Analysis**:

   The parsed data is passed to the **analysis module**, which performs advanced analysis to identify trends, anomalies, or performance degradation metrics.

5. **Graph Generation**:

   The analyzed data is forwarded to the **visualization module**, where graphical insights like error trends and warnings are generated using GNUPLOT.

6. **Results Delivery**:

   Finally, the generated insights, which include both text-based summaries and visual graphs, are delivered back to the CLI for the user to review. The user can now examine the analysis results, understand the trends and issues, and make informed decisions about the next steps, whether it be addressing errors, optimizing performance, or monitoring system health.

The generated insights, including text-based summaries and visual graphs, are returned to the CLI for the user to review. The user can analyze the results to understand trends, errors, and performance issues. Based on this information, the user can make informed decisions to address problems, optimize performance, or monitor system health.

3.3 Use Case Diagram:

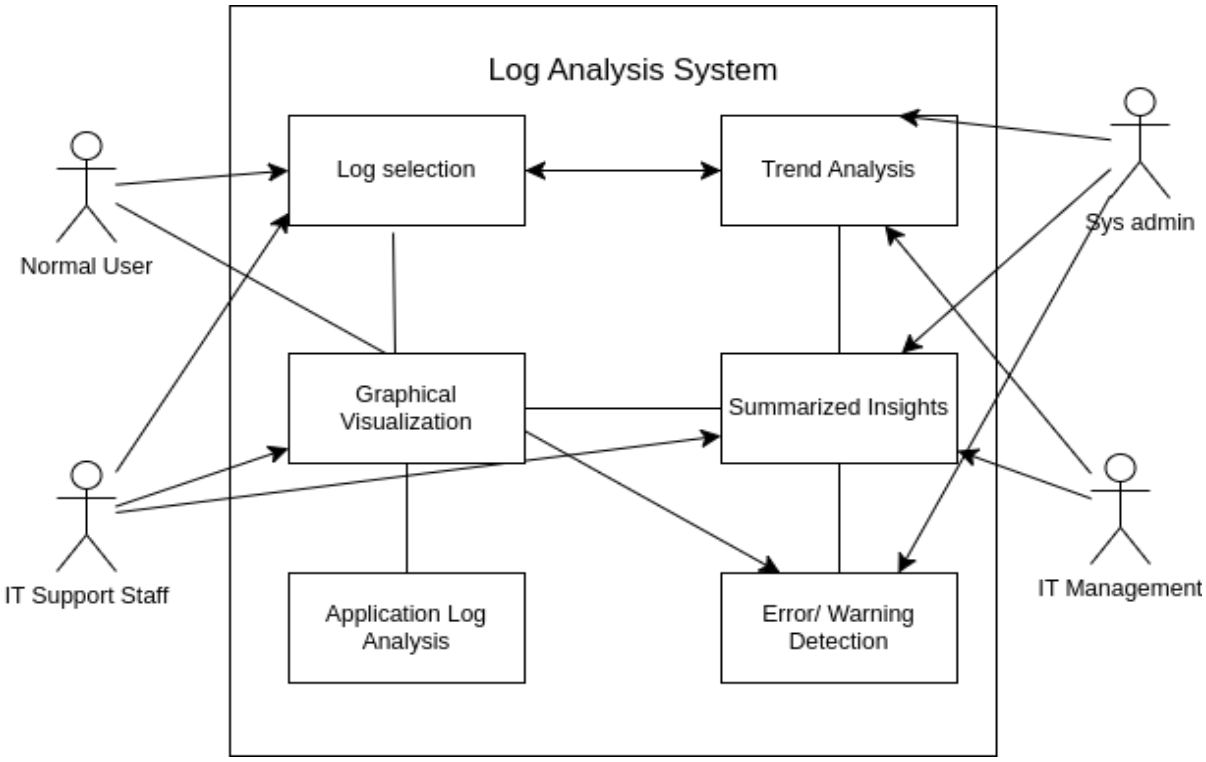


Fig 3.3 Different Use Cases of different users

The **Use Case Diagram** provides a high-level overview of the interactions between the actors (users) and the system. It identifies the primary use cases of the log analysis system and maps them to their respective stakeholders.

Components:

1. **Actors**:

- **IT Support Staff**: The primary user responsible for initiating log analysis, reviewing results, and generating reports.
- **System Administrator**: Handles technical use cases like configuring the system for advanced log parsing or debugging errors.

- **IT Management**: Focused on reviewing summarised insights and trends for decision-making.

2. **System**:

The Log Analysis System acts as a boundary that encapsulates all use cases related to log retrieval, parsing, analysis, and visualization. It defines the scope of operations, ensuring that all processes involved in log handling, from data collection to insight generation, are managed within this system.

3. **Use Cases**:

- **Log Selection**:

  Users can choose specific logs for analysis based on criteria like date, application, or event type. This feature helps users focus on relevant log files to troubleshoot or monitor system health.

- **Trend Analysis**:

  The system identifies patterns in log data, such as error frequency over time, to detect recurring issues. Trend analysis helps users understand system behavior and pinpoint areas needing attention.

- **Graphical Visualisation**:

  Using tools like GNUPLOT, the system creates visual representations of trends such as error counts. Graphs make it easier to identify patterns and make informed decisions.

- **Summarized Insights:**

  The tool generates high-level summaries of the log analysis, highlighting key findings like errors or performance issues. These summaries provide users with a quick overview of potential system problems.

- **Application-Specific Log Analysis**:

  Users can focus on logs related to a particular application, isolating specific issues. This enables faster identification and resolution of application-related problems.

- **Error and Warning Detection:**

  The system automatically identifies critical errors and warnings in the logs. This ensures that important issues are flagged for prompt action to avoid system disruptions.

- **Anomaly Detection**:

  The tool detects unusual patterns in logs, such as unexpected spikes or abnormal behaviors. Anomaly detection helps identify security threats or misconfigurations early on.

- **Performance Monitoring:**

  The system tracks performance metrics like CPU usage and memory consumption through log analysis. It helps users monitor system health and take proactive steps to maintain performance.

4. **Interactions**:

- **IT Support Staff**

  IT Support Staff interacts with a majority of the use cases, such as analyzing logs, detecting errors, and generating visual graphs for trend analysis. They from the system can reviews logs to troubleshoot issues, monitor system health, and see operations details. Their role involves using the tool to quickly identify and resolve common system problems

- **System Administrator**

  System Administrators can use the log analysis system to read detailed logs and check for issues like login failures or system errors. They can track user activities, monitor system performance, and identify any unusual behavior or security concerns.

- **IT Management**

  They relies on Summarized Insights and Trend Analysis to make data-driven decisions about system health and future investments. By reviewing high-level summaries of

performance and error trends, they can identify areas that need improvement or additional resources.

Diagram Structure:

- Place actors (**stick figures**) outside the system boundary (**rectangle**).
- Add **use cases** (**ovals**) inside the rectangle, grouped by functionality.
- Draw **lines** connecting actors to relevant use cases, representing interaction.

This diagram helps stakeholders quickly understand who uses the system, how they use it, and what functionalities the system provides. It also ensures clarity in roles and responsibilities for each feature.

4. Project Evolution

4.1 Phase - I: Planning and Methodology Development

In the first phase of our project, we focused on defining the scope, identifying key objectives, and outlining the methodology for automating log file analysis. This initial phase was crucial for setting a strong foundation for the development of the Log File Analyzer, ensuring a clear understanding of the project's goals and the steps needed to achieve them.

4.1.1. Project Scope and Focus

We established the primary objective of automating the analysis of system and application logs using shell scripting in a UNIX-based environment. The SHELL program was conceptualised to efficiently parse large volumes of log data to identify critical events, errors, warnings, and process-related information. Key considerations included:

- The types of logs to be analyzed (e.g., syslog, auth.log, dpkg log).
- The time period of logs to focus on for analysis.
- Tools and techniques to be employed for log parsing and analysis.
- Boundaries and potential limitations, such as handling log file formats specific to certain operating systems.

4.1.2. Methodology

We developed a methodology that emphasised automation, efficiency, and user-centric design. The plan involved using shell scripting to extract and process key metrics from log files, ensuring that administrators could gain actionable insights without manual intervention. The methodology incorporated the use of shell utilities such as grep, awk, sed, and others was proposed for log

parsing and data manipulation. These tools were chosen for their ability to handle large log files efficiently and their compatibility with UNIX-based systems.

4.1.3. Proposed Key Metrics

We identified several key metrics to focus on during the analysis phase. While these were not defined exhaustively at this stage, we outlined the following primary areas of interest:

- **Errors**:
  - Identification of critical errors, such as server crashes, segmentation faults, or configuration issues, recorded in logs.
  - Extraction of HTTP 500-series errors from web server logs and system-critical errors from system logs.
  - Summaries of error frequency, time of occurrence, and affected systems or applications.

- **Warnings**:
  - Capture of warning messages indicating potential issues, such as high memory usage or low disk space.
  - Pattern recognition to identify trends in warning occurrences over time.

- **Process Information**:
  - Analysis of process start/stop events and their impact on system performance.
  - Tracking of failed login attempts for security monitoring.
  - Monitoring of unexpected process terminations to detect system instability.

- **Performance Metrics**:

  Extraction of CPU load, memory usage, and disk I/O data from system logs to identify performance bottlenecks.

By focusing on these metrics, we aimed to design a solution that could provide a comprehensive overview of system health and security risks, enabling timely and effective responses.

4.1.4. Outcome of Phase 1

The outcome of this phase was a well-defined methodology and a clear vision for the Log File Analyzer. Key deliverables included:

- A detailed mapping of the proposed workflow for log analysis.
- A draft of the menu-driven interface to guide user interaction.
- An initial list of metrics and focus areas for log analysis, providing direction for the subsequent phases of development.

Phase 1 served as the blueprint for the project, ensuring that the subsequent developent stages were aligned with the project's goals and objectives.

4.2 Phase II: Implementation

In the second phase of the project, we developed and implemented four major shell scripts to analyze different Ubuntu log files. Each script was designed to focus on specific system metrics, errors, and activities, enabling comprehensive log analysis. Below is an overview of the work performed under each script, detailing the key focus areas and insights derived.

4.2.1. test_mem.sh: Memory Log Analysis

This script was designed to analyze memory-related log entries and extract critical information related to system resource utilization and memory errors.

- **Error Categorization**:
    - **Severe**: Out-of-Memory (OOM) errors, segmentation faults.
    - **Mild**: Minor memory allocation issues.
    - **Warnings**: Resource usage nearing limits, potential system bottlenecks.

- **System Resource Monitoring**:
    - **Memory Information**: Total, used, free, and cached memory statistics.
    - **Disk Usage**: Available and used disk space to monitor storage capacity.
    - **Swap Space**: Total, used, and free swap memory.

- **Analysis Insights**:
    - Identified patterns of memory usage to predict and prevent resource exhaustion.
    - Provided actionable insights into system stability by highlighting frequent memory-related errors.

4.2.2. test_proc.sh: Process Log Analysis

This script focused on monitoring and analyzing logs for system processes to identify performance and stability issues.

- **Process Error Tracking**:
    - **Crashes**: Unexpected process terminations identified by crash-related keywords.
    - **Hangs**: Non-responsive processes tracked using patterns like "timeout" or "not responding."
    - **Forced Terminations**: Logs indicating manual or system-initiated process terminations (e.g., SIGKILL).

- **CPU and GPU Monitoring**:
    - Tracked CPU-related issues such as overheating, throttling, or excessive load.
    - Identified GPU crashes or memory overflows, providing insights into graphical or compute-intensive workloads.

- **Analysis Insights**:
    - Highlighted critical processes prone to instability.
    - Provided an organized summary of error types to prioritize resolution.

4.2.3. test_dpkg.sh: Package Management Log Analysis

This script analyzed the dpkg.log file to monitor software management activities and provide summaries of system changes.

- **Key Actions Tracked**:
    - **Installations**: Logged new package installations and their versions.
    - **Upgrades**: Captured updates to existing packages, ensuring system updates were monitored.
    - **Removals**: Identified package removals to detect potential dependency issues.
    - **Configurations**: Tracked the configuration of packages for proper functionality.
    - **Unpacking**: Logged intermediate steps during installation or upgrade processes.

- **Insights Provided**:
    - Detected patterns of frequent package removals, signaling potential misconfigurations.
    - Summarized changes in software state over a given period for easy tracking.

4.2.4. test_auth.sh: Authentication Log Analysis

This script focused on parsing authentication logs (auth.log) to monitor system access and detect potential security risks.

- **Event Categorization**:
    - **Session Management**: Tracked session openings and closings for each user.
    - **Authentication Attempts**: Identified successful and failed login attempts, including brute-force attempts.
    - **System Security Events**: Monitored actions from processes like sshd and sudo for unauthorized or suspicious activities.

- **Insights Provided**:
    - Highlighted users with frequent failed login attempts, signaling potential intrusions.
    - Tracked legitimate user access patterns for compliance and auditing.
    - Detected anomalies in system authentication, improving overall security posture.

4.2.5 Outcome of Phase 2

Through these scripts, we implemented a structured and automated approach to analyzing key aspects of system performance, stability, and security. This phase established the groundwork for detailed log parsing, error categorization, and resource monitoring, empowering administrators to identify and address critical issues efficiently. Each script added significant value by providing actionable insights into specific areas of the system, making this phase a crucial milestone in the project's development.

4.3 Final Phase: Extension to Multiple Operating Systems

One of the key features introduced in this final phase is the ability for users to provide **custom log files**. Previously, the tool was designed to work only with the default system log files on Ubuntu. However, in recognition of the fact that users may want to analyze logs from other systems or specific applications, we enabled the feature to accept user-provided log files in various formats. This means that users can now input their own log files, regardless of the operating system they are working on.

By offering custom log file support, the tool becomes more flexible, allowing it to analyze logs generated from applications or systems outside the typical scope of Ubuntu’s default logs. This can be particularly useful for developers or system administrators who need to debug specific applications or services that log errors in non-standard formats. The user simply needs to provide the log file path, and the tool will process the log entries accordingly.

Additionally , we significantly expanded its scope by extending compatibility to **multiple operating systems**, including Windows, macOS, and Solaris. This extension aimed to ensure our solution's versatility across diverse environments, addressing platform-specific requirements and error-handling mechanisms.

4.3.1. Solaris

This section focuses on the Solaris-specific script, which exemplifies our methodology for detecting and analyzing errors on the platform. Below, we detail the types of errors being analyzed and categorized, followed by a discussion of the logic employed to ensure accurate and comprehensive logging and reporting.

4.3.1.1. General Error Categories

- **Severe Errors**
    - Indicate critical failures or issues that might require immediate attention.
      Examples include:
        - Kernel bugs.
        - Out of Memory (OOM) errors.
        - Disk failures and I/O errors.
        - Process crashes, hangs, or terminations.
        - CPU or GPU faults.
        - PAM (Pluggable Authentication Module) errors.
- **Mild Errors**
    - Less critical but still important to monitor:
        - Generic errors or failures (e.g., login issues, CPU/GPU warnings).
        - Connection issues, such as connection refused.
- **Warnings**
    - Indicate potential problems or deprecated usage:
        - High usage or performance degradation.
        - Repeated login failures.
        - Warnings related to system resources.

- **Success Messages**
    - Default category for logs that do not indicate errors or warnings.

4.3.1.2. Solaris-Specific Error Analysis

- **PAM Errors**
  - Analyzes PAM-related issues (e.g., PAM_ERROR_MSG) for authentication failures.
- **Login Failures**
  - Tracks repeated login failures, which may signal brute force attacks or misconfigured access.
- **Account Disabled Issues**
  - Monitors errors indicating that user accounts are disabled.

4.3.1.3. Connection Issues

- Tracks problems with network connectivity, including:
  - Refused connections.
  - Unauthorized access attempts.

4.3.1.4. Memory and Resource Management

- **Out of Memory (OOM) Errors**
  - Tracks logs mentioning OOM or "out of memory" explicitly.
- **Malloc Errors**
  - Monitors memory allocation issues (malloc failures).
- **Segmentation Faults (Segfaults)**
  - Detects segfault messages to identify invalid memory accesses.

4.3.1.5. Processor and GPU Issues

- **CPU Errors**
  - Tracks processor-specific issues like CPU overloads or faults.
- **GPU Errors**
  - Monitors GPU-related failures such as overload or hardware faults.

4.3.1.6. File Transfer Protocol (FTP) Monitoring

- Analyzes FTP transactions:
    - FTPD: IMPORT
    - FTPD: EXPORT
- Tracks usage patterns and identifies anomalies.

4.3.1.7. Dynamic Log Parsing

- The script dynamically adjusts to different log date formats (e.g., yyyy-mm-dd or mmm dd) to process logs correctly based on the provided date range.

4.3.1.8. Summary and Visualization

- The script generates:
    - A summary table of error counts per application.
    - A graph visualizing the distribution of severe errors, mild errors, warnings, and successes.

4.3.2.MacOs

This macOS-specific script focuses on log parsing and error categorization, ensuring a comprehensive understanding of system performance and potential failures. Below is a breakdown of the key areas analyzed:

4.3.2.1. Error Categorization

- **Severe Errors:** Include critical issues such as kernel panics, process crashes, disk failures, and resource allocation errors.
- **Mild Errors:** Cover less severe issues like application errors, memory warnings, or minor process issues.
- **Warnings:** Encompass deprecated features, low disk space, and other non-critical advisories.
- **Success:** Tracks successful operations to balance error reporting.

4.3.2.2. Specific Error Tracking

The script meticulously tracks various types of system errors, categorized into specific groups for deeper analysis:

- **Memory Issues:**
  - Out-of-Memory (OOM) errors.
  - Memory allocation failures (e.g., malloc errors).
- **Process Failures:**

  This part handles various process-related errors, categorized into three types:

  Process Crash Errors

  - Key Error: "process crash"
  - Analysis: Identifies when processes crash and lists them with the app name, count, and error message ("process crash").

  Process Hang Errors

  - Key Error: "process hang"
  - Analysis: Detects when processes hang (i.e., become unresponsive), categorizing them with app name, count, and message ("process hang").

  Process Killed Errors

  - Key Error: "process killed"
  - Analysis: Monitors when processes are killed, displaying the app name, error count, and message ("process killed").

- **Kernel Issues:**
  - Kernel panics and crashes (EXC_BAD_ACCESS, EXC_CRASH).
- **Disk I/O Errors:**
  - Disk failures and input/output read/write issues.

- **Hardware Issues:**

  CPU and GPU faults, overloads, or panics.

4.3.3.Windows

4.3.3.1. Error Categorization

- **Severe Errors**: Include critical issues such as kernel panics, process crashes, disk failures, and resource allocation errors.
- **Mild Errors**: Cover less severe issues like application errors, memory warnings, or minor process issues.
- **Warnings**: Encompass deprecated features, low disk space, and other non-critical advisories.
- **Success**: Tracks successful operations to balance error reporting.

4.3.3.2. Specific Errors

The following are the specific error categories being tracked based on messages from system logs or application outputs:

1. **Process Crash Errors**
   - When an application process unexpectedly crashes, it is recorded under the **Process Crash Errors** category.

2. **Process Hang Error**

   - These errors are triggered when a process becomes unresponsive, and no longer performs any actions, indicating a potential issue with the application or system resources.

3. **Process Killed Errors**

   - These errors are recorded when a process is killed by the system or an external source, which could be due to resource limitations or a critical failure.

4. **CPU Errors**
    - **CPU Fault, CPU Overload, CPU Failure**: These errors point to potential issues with the central processing unit, affecting overall system performance or stability.
5. **GPU Errors**
    - **GPU Failure, GPU Overload, GPU Fault**: These errors are related to graphical processing units and can affect rendering applications or games, especially those relying heavily on GPU resources.
6. **OOM (Out of Memory) Errors**
    - Occurs when the system or application runs out of available memory, leading to potential application crashes or system instability.
7. **Segfault (Segmentation Fault) Errors**
    - Occurs when an application tries to access restricted memory areas, causing a segmentation fault and crashing the program.
8. **Malloc Errors**
    - Memory allocation failures that can lead to errors in applications that need to allocate memory dynamically.
9. **Kernel Errors**
    - These errors represent issues within the kernel, which could cause system crashes, blue screens, or other system-wide failures.
10. **Critical Errors**
    - A category for other critical errors that can destabilize the system or application, requiring immediate attention
11. **Disk Failure / I/O Errors**
    - Indicates problems with the disk drives or Input/Output operations that may affect the stability of file access and system responsiveness.

5. Limitations

While our project is designed to offer valuable insights into system errors, it comes with certain limitations that must be considered when using the tool. These limitations are outlined below:

5.1. Platform Dependency: Ubuntu Shell

Currently, the project is designed and optimized to run only on an Ubuntu shell. This limitation arises from the reliance on shell commands and tools that are specific to the Ubuntu environment. If the project is executed on a different terminal, such as on other Linux distributions (e.g., CentOS, Fedora) or operating systems (e.g., macOS, Windows), there are likely to be compatibility issues. This is because different environments may have different versions of shell utilities, varying syntax for commands, or completely different command-line tools.

For instance, tools like dmesg, journalctl, or certain file paths used to access system logs may not function identically or even exist in other systems. As a result, running the project outside of Ubuntu can cause errors in the log data retrieval process, and it may fail to generate the correct analysis. To overcome this limitation, the project would need significant re-engineering to support cross-platform compatibility, ensuring that system logs and error handling mechanisms are uniform across different environments.

5.2 Log Data Analysis : Time Constraints

Another key limitation is related to the log file retention policy. In many Linux systems, including Ubuntu, system logs are compressed after a certain period to save disk space. Typically, logs older than one week are compressed into archives such as .gz files. This presents a challenge for our project, as it can only analyze logs from the most recent week. Once the logs are compressed, our project is unable to access or process these older entries without additional preprocessing to decompress them.

This means that any system errors or events that occurred more than a week ago are effectively inaccessible for analysis. This limitation can be particularly problematic if an error of interest happened more than a week ago but was not captured by real-time monitoring. To mitigate this

issue, the project could be modified to include a decompression step or be configured to retain logs for a longer period, but this would require changes to the system log configuration.

5.3. Dependency on Log File Integrity

The project heavily relies on the integrity of the system log files for error analysis. If there are any issues with the log files—such as corruption, misconfiguration, or missing entries—the project will not be able to perform an accurate analysis. In such cases, the errors might not be logged properly, or the log data may be incomplete, resulting in incorrect or misleading analysis outputs.

For example, if the system is configured to rotate logs improperly or the log file is truncated or overwritten, the project will either fail to capture the necessary information or produce incomplete reports. Additionally, if the logs are tampered with or contain erroneous data, the analysis could be skewed, leading to incorrect conclusions. The tool does not currently have built-in error handling for such issues, and therefore it assumes that the log files are intact and properly formatted. A possible improvement would be to implement robust error-handling mechanisms to check the validity of the logs before analyzing them, as well as to provide feedback to users in case of corruption or other issues.

6. Team Member Details and Their Contribution

The successful completion of this project was made possible by the collaborative efforts of the following team members, each bringing their expertise to different aspects of the system:

1. Shubham Devangan and Aryan Prajapati: Log Analysis on Ubuntu and Solarais

Shubham Devangan and Aryan Prajapati collaboratively worked on an extensive log analysis project, focusing on identifying, extracting, and analyzing critical fields across multiple log files. These included `syslog`, `auth.log`, `systat`, and `dpkg.log`. The efforts were aimed at creating a comprehensive system capable of analyzing and visualizing key metrics while providing a user-friendly interface. Below are the key actions we performed during the project:

Log File Identification and Usage:

- We identified critical log files, including syslog, auth.log, systat, and dpkg.log, for comprehensive analysis.
- Synthesized SolarisOS logs using standard formats and metrics due to the lack of official logs.
- Ensured accurate extraction of meaningful insights by understanding each log's structure and data.

Analysis with syslog:

- Memory Metrics: Analyzed memory usage metrics such as page-ins, page-outs, and free memory to classify errors into severe, mild, or warnings.
- Process Metrics: Monitored key process-related indicators like CPU usage, segmentation faults, and kernel panics to identify error types efficiently.
- Actionable Insights: Generated insights to address memory leaks, process crashes, and resource bottlenecks by analyzing memory consumption and process crashes.

Analysis with auth.log:

- Failed Login Metrics: Tracked timestamps of failed login attempts and monitored the authentication failure rate to detect potential security threats.
- Sudo Command Metrics: Monitored and counted sudo command usage to identify anomalies, such as unauthorized privilege escalation attempts.
- Security Insights: Improved system security by detecting abnormal authentication behaviors, such as repeated failed login attempts or elevated privileges used unexpectedly.

Analysis with dpkg.log:

- Package Installation Metrics: Extracted details of package changes, such as installation timestamps, package names, and status codes (e.g., success, failure, partial installation).

- Package Operation Metrics: Analyzed "when" packages were installed, removed, or updated to track system changes and potential issues.
- Configuration Insights: Identified misconfigurations and tracked package dependencies to ensure system stability and integrity based on package log data.

Analysis with systat:

- **Memory Usage Metrics**: Collected detailed memory statistics, including page-ins, page-outs, and memory utilization percentage, to monitor daily usage patterns.
- **Resource Monitoring Metrics**: Displayed real-time memory metrics in a CLI interface, showing key values such as used memory, cached memory, and swap usage for efficient monitoring.
- **Resource Management Insights:** Improved system resource management by identifying overutilization and memory spikes, leading to optimized memory allocation.

Custom Log Support and Debugging:

- **Log Format Support**: Created an interface that allows users to upload and analyze custom log files, supporting various formats from Ubuntu 22.04 and 24.04.
- **Debugging Metrics:** Integrated debugging features, such as error logs, stack traces, and debug messages, to ensure seamless operation and efficient error resolution.
- **Flexible Log Handling**: Provided support for different log formats, allowing for smooth analysis of non-standard logs and error reports across multiple versions.

SolarisOS Log Synthesis and Analysis:

- **Log Generation Metrics**: Designed custom log generators based on Solaris standard log formats and error patterns like out-of-memory errors, disk failures, and kernel crashes.

- **Error Classification**: Conducted a thorough analysis of synthesized logs to classify errors into categories such as severe, mild, and warning, based on key metrics.
- **Visualization Insights**: Visualized error trends, such as application crashes, resource utilization spikes, and warning logs, using graphical representations to provide actionable insightThrough these efforts, we created a robust system capable of analyzing diverse log files, supporting cross-platform compatibility, and offering insights in an intuitive and visually engaging manner

2. Lakshay Baijal: Windows Log File Analysis

Lakshay spearheaded the extension of the project to include the analysis of Windows log files, broadening the tool's compatibility and applicability. His contributions encompassed:

Windows Log File Analysis:

- Adapted the system to process Windows Event Logs, focusing on critical types such as System, Application, and Component-Based Servicing (CBS) Logs.
- Mapped Windows-specific error categories to a standardized format, enabling seamless integration with the existing analysis framework.
- Conducted detailed analysis to extract insights on process crashes, CPU/GPU faults, disk failures, and out-of-memory (OOM) errors from System and Application Logs.
- Analyzed CBS Logs to identify misconfigurations, failed updates, and package-related errors critical for system integrity.

Enhanced Visualization:

- Integrated Gnuplot to create detailed graphs representing trends like error frequency and resource utilization.
- Enabled graph export in user-friendly formats for presentations and reports, enhancing the accessibility and clarity of insights.
- Provided clear visualizations that empowered users to detect patterns and address issues effectively, making the system a valuable tool for Windows environments.

Lakshay's work significantly enhanced the tool's functionality, enabling it to seamlessly handle Windows log formats while offering improved visualization capabilities, thus expanding its usability across diverse operating systems.

4.Vanshika Singh: Log Analysis on MacOS

Vanshika played a crucial role in extending the system's functionality to support macOS, addressing its unique log structures and error reporting mechanisms. Her contributions included:

- **Comprehensive Log Analysis**:
    - Analyzed macOS system logs to identify and track common errors such as **process hangs**, **memory leaks**, and **disk I/O failures**.
    - Customized the log parsing engine to handle macOS-specific log formats, ensuring accurate extraction of critical insights like timestamps, error codes, and system metrics.
- **Performance Monitoring**:
    - Implemented modules to monitor macOS resource usage, such as **CPU and memory utilization**, enabling the detection of performance bottlenecks and optimization opportunities.
    - Designed a system to track and categorize recurring issues, providing insights into the severity and frequency of problems.

- **Error Reporting and Debugging**:

    - Developed detailed error classification using pattern matching tailored to macOS logs, enabling clear categorization of issues by severity (e.g., minor, critical).
    - Enhanced debugging features to streamline the identification and resolution of errors specific to macOS applications and system processes.

Vanshika's efforts ensured that the tool effectively handled macOS-specific challenges, making it an indispensable resource for system administrators and developers working in macOS environments.

7. Result

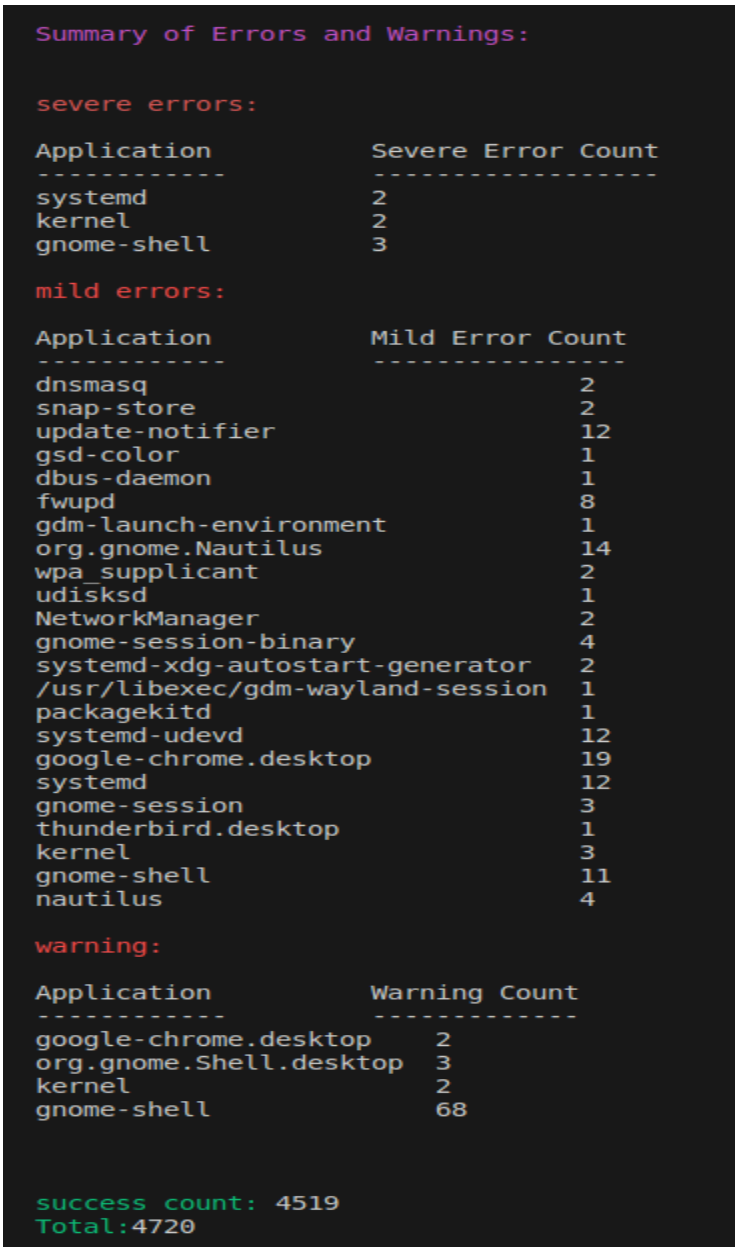
The Log File Analyzer simplified log analysis by providing a user-friendly interface for quickly identifying and categorizing errors as severe, mild, or warning. It offered insights into system performance, memory usage, authentication issues, and package changes, enhancing both system monitoring and security. Supporting custom log files with flexible regex patterns, the tool automated error detection and reporting, allowing users to prioritize troubleshooting and improve efficiency. The details of results are shown in Appendix .The result was a more effective and time-saving approach to log management and system oversight.

References

[1]. Source Code : https://github.com/LakshayBaijal/SSD_Project_Team30/tree/final

[2]. Video : https://github.com/LakshayBaijal/SSD_Project_Team30/tree/final/readme

Appendix:



```
Summary of Errors and Warnings:


severe errors:

Application           Severe Error Count
------------          ------------------
systemd               2
kernel                2
gnome-shell           3

mild errors:

Application           Mild Error Count
------------          ---------------
dnsmasq                             2
snap-store                          2
update-notifier                     12
gsd-color                           1
dbus-daemon                         1
fwupd                               8
gdm-launch-environment              1
org.gnome.Nautilus                  14
wpa_supplicant                      2
udisksd                             1
NetworkManager                      2
gnome-session-binary                4
systemd-xdg-autostart-generator     2
/usr/libexec/gdm-wayland-session    1
packagekitd                         1
systemd-udevd                       12
google-chrome.desktop               19
systemd                             12
gnome-session                       3
thunderbird.desktop                 1
kernel                              3
gnome-shell                         11
nautilus                            4

warning:

Application           Warning Count
------------          -------------
google-chrome.desktop     2
org.gnome.Shell.desktop   3
kernel                    2
gnome-shell               68



success count: 4519
Total:4720
```

Fig A.1 Ubuntu Memory log analysis

```
Graph:

Severe Errors:  7
Mild Errors:    119
Warnings:       75
Others:         4519

Error Table:

Error               App Name        Count      Error Message
oom                 systemd         2          oom
oom                 kernel          2          oom
kernel bug          gnome-shell     3          kernel bug
segfault            N/A             0          segfault
malloc              N/A             0          malloc
critical            N/A             0          critical
disk failure        N/A             0          disk failure / I/O error

Generated graph: graphs/2024-11-25/error_counts.png
```

Fig A.2 Ubuntu Memory log analysis graph

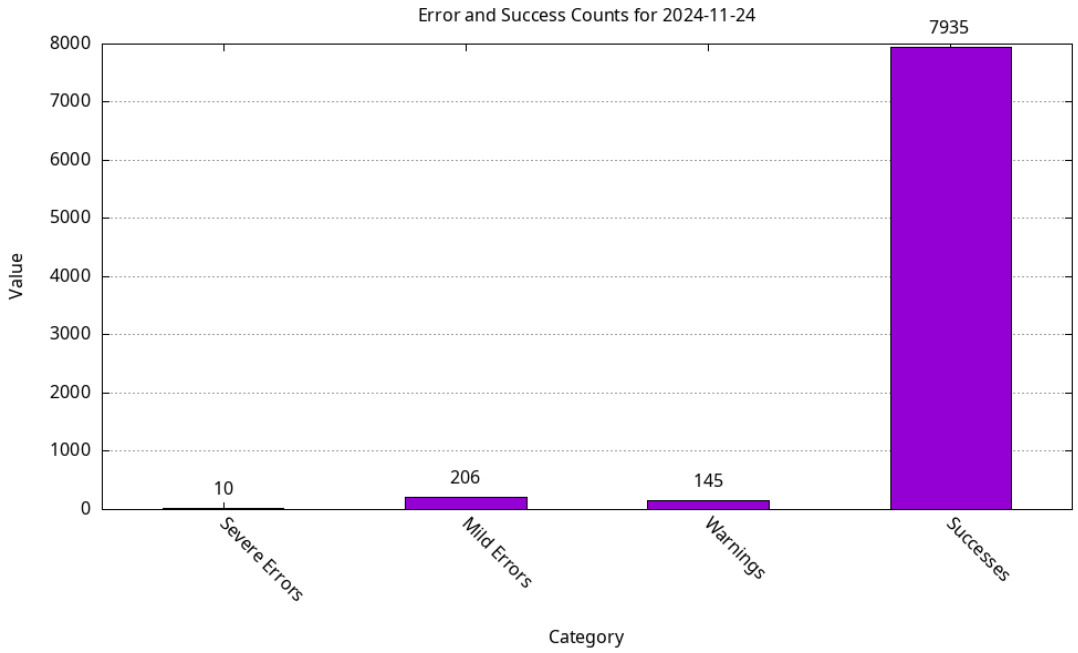


Fig A.3 Ubuntu memory analysis graph

```
Summary of Errors and Warnings:

Error Type           App Name                          Count
-------------------  --------------                    -----
Severe Error         gnome-shell                       3

Error Type           App Name                          Count
-------------------  --------------                    -----
Mild Error           google-chrome.desktop             19
Mild Error           org.gnome.Nautilus                14
Mild Error           update-notifier                   12
Mild Error           systemd-udevd                     12
Mild Error           systemd                           12
Mild Error           gnome-shell                       11
Mild Error           fwupd                             8
Mild Error           nautilus                          4
Mild Error           gnome-session-binary              4
Mild Error           kernel                            3
Mild Error           gnome-session                     3
Mild Error           wpa_supplicant                    2
Mild Error           systemd-xdg-autostart-generator   2
Mild Error           snap-store                        2
Mild Error           NetworkManager                    2
Mild Error           dnsmasq                           2
Mild Error           /usr/libexec/gdm-wayland-session  1
Mild Error           udisksd                           1
Mild Error           thunderbird.desktop               1
Mild Error           packagekitd                       1
Mild Error           gsd-color                         1
Mild Error           gdm-launch-environment            1
Mild Error           dbus-daemon                       1

Error Type           App Name                          Count
-------------------  --------------                    -----
Warning              gnome-shell                       68
Warning              org.gnome.Shell.desktop           3
Warning              kernel                            2
Warning              google-chrome.desktop             2
```

Fig A.4 Ubuntu syslog analysis for process

```
Debug Info:

Total Severe Errors: 3
Total Mild Errors: 119
Total Warnings: 75


success count: 4527
Total:4724



Process-Related Log Graph:

Severe Errors:  3
Mild Errors:    119
Warnings:       75
Total Success:  4527


Error Table:

Error              App Name       Count     Error Message
process killed     gnome-shell    3         process killed
process crash      N/A            0         process crash
process hang       N/A            0         process hang
cpu error          N/A            0         cpu fault
gpu error          N/A            0         gpu failure
```

Fig A.5 Ubuntu syslog analysis for process (graph on shell)

```
Authentication Failures
Timestamp                User                    Logname
---------                ----                    -------
21:16:59                 aryanprajapati          aryanprajapati

Sudo Privileges
Timestamp                Sudo User               Command
---------                ---------               -----
01:31:44                 root                    /usr/lib/update-notifier/package-system-locked

06:07:42                 root                    /usr/lib/update-notifier/package-system-locked

09:15:37                 root                    /usr/lib/update-notifier/package-system-locked

20:10:35                 root                    /usr/lib/update-notifier/package-system-locked

20:34:30                 root                    /usr/lib/update-notifier/package-system-locked


Session Report
Application Name         Number of Sessions
----------------         ------------------
pkexec                   5
su                       3
gdm-password]            3
gdm-launch-environment]  3
CRON                     118
(systemd)                7
```

Fig A.6 Ubuntu auth log file analysis

```
Graphical Representation:

Failed Logins:      1
Successful Logins:  5
Session Openings:   139
Other Events:       251

Total Events:       396
```

Fig A.7 Ubuntu Auth log file analysis graph on shell

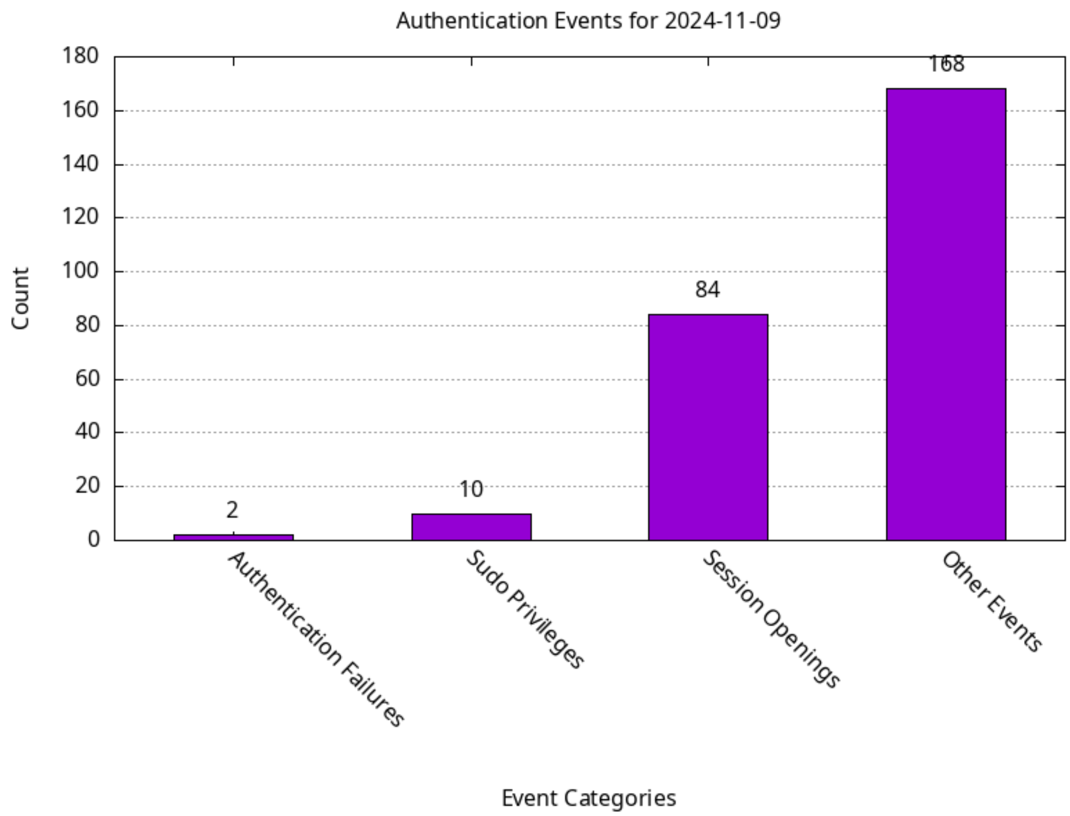


Fig A.8 Ubuntu Auth log file result graph

```
Package Report

Date                 Timestamp            Package Name                    Action Performed
--------             -----------          ------------                    ----------------
2024-11-29           06:07:27             libsoup-2.4-1:amd64             configure
2024-11-29           06:07:30             gir1.2-soup-3.0:amd64           configure
2024-11-29           06:07:27             libsoup2.4-common:all           configure
2024-11-29           06:07:30             libsoup-3.0-common:all          configure
2024-11-29           06:07:30             libsoup-3.0-0:amd64             configure


Graphical Representation of Actions:
configure      :                                              5

Total Package Actions: 5
```

Fig A.9 Ubuntu Package manager log results

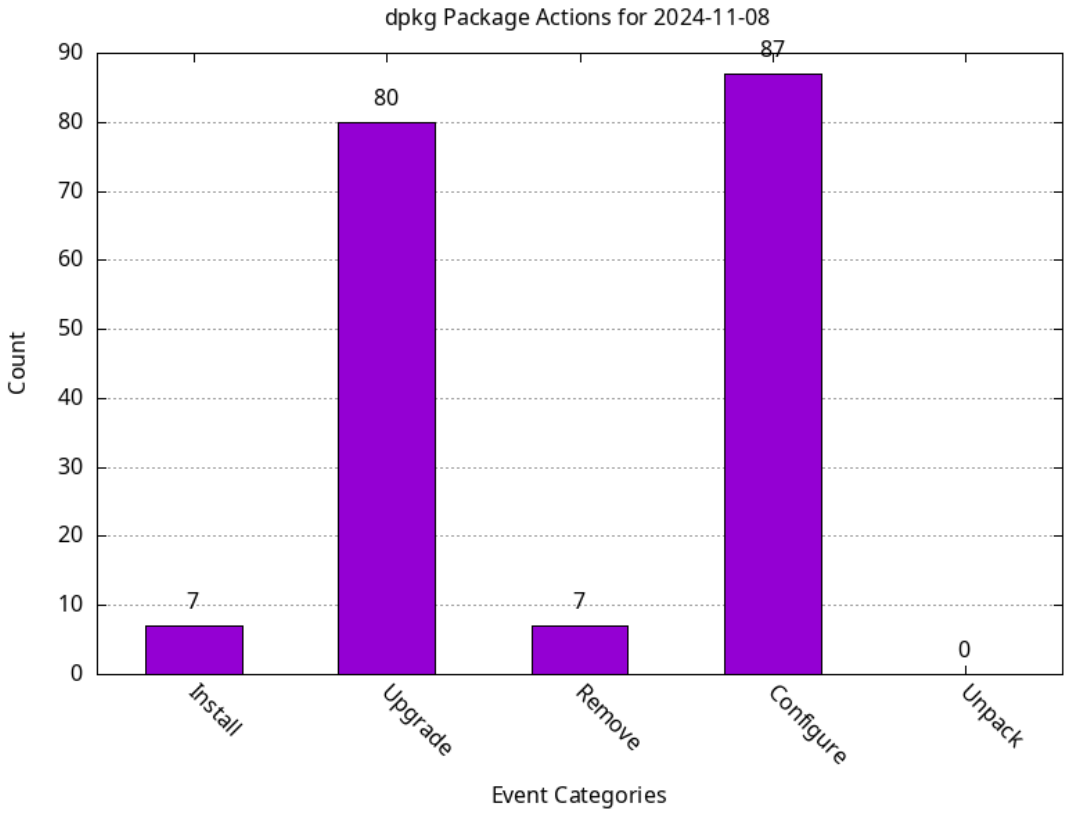


Fig A.10 Ubuntu Package manager log analysis graph

```
High memory usages process are:

aryanpr+ 6.1 /snap/code/176/usr/share/code/code
aryanpr+ 5.6 /opt/google/chrome/chrome
mysql 4.6 /usr/sbin/mysqld
aryanpr+ 4.6 /opt/google/chrome/chrome
aryanpr+ 4.4 /snap/code/176/usr/share/code/code
aryanpr+ 3.7 /usr/bin/gnome-shell
aryanpr+ 3.4 /opt/google/chrome/chrome
aryanpr+ 3.3 /snap/code/176/usr/share/code/code
aryanpr+ 3.1 /usr/bin/nautilus
aryanpr+ 2.9 /snap/code/176/usr/share/code/code

Memory and paging details for interval


Displaying system stats for 2024-11-28 00:00:00 to 2024-11-28 23:00:00

Linux 6.8.0-49-generic (aryanprajapati-Dell-G15-5520)   11/28/2024      _x86_64_        (16 CPU)
```

Fig A.11 Ubuntu stat file analysis result

01:40:03 AM	pgpgin/s	pgpgout/s	fault/s	majflt/s	pgfree/s	pgscank/s	pgscand/s	pgsteal/s	%vmeff
02:10:07 AM	9.99	4.02	9.89	0.05	111.65	0.00	0.00	0.00	0.0%
02:40:07 AM	257.87	18.33	33.78	0.00	83.24	0.00	0.00	0.00	0.0%
03:10:07 AM	2.28	2.37	12.20	0.01	26.80	0.00	0.00	0.00	0.0%
03:40:05 AM	0.00	2.03	3.58	0.00	20.51	0.00	0.00	0.00	0.0%
04:10:07 AM	0.00	1.78	3.19	0.00	16.98	0.00	0.00	0.00	0.0%
04:40:07 AM	0.00	2.76	4.86	0.00	43.05	0.00	0.00	0.00	0.0%
05:10:07 AM	2.36	4.15	6.29	0.00	43.68	0.00	0.00	0.00	0.0%
05:40:07 AM	360.62	112.40	1924.07	1.63	9289.46	0.00	0.00	0.00	0.0%
06:10:07 AM	57.29	129.26	1456.52	0.05	4824.95	0.00	0.00	0.00	0.0%
06:40:03 AM	176.60	1062.85	11474.38	0.14	24477.38	174.95	0.14	338.72	193.5%
07:10:07 AM	5.40	95.73	15296.71	0.00	38571.36	0.00	0.00	0.00	0.0%
07:40:07 AM	126.60	162.67	10211.75	0.08	18148.17	102.46	0.05	176.68	172.3%
08:29:41 AM	10.74	8.59	82.17	0.05	136.63	0.00	0.00	0.00	0.0%
08:40:07 AM	174.26	372.12	27114.18	0.28	45897.32	0.00	0.00	0.00	0.0%
09:10:07 AM	174.69	247.03	15188.12	1.04	24036.01	0.00	0.00	0.00	0.0%
09:40:07 AM	258.50	291.73	1179.14	0.21	5327.22	37.43	0.00	66.14	176.7%
10:10:07 AM	0.37	47.95	224.21	0.00	946.55	0.00	0.00	0.00	0.0%
10:40:02 AM	43.19	278.07	2507.54	0.31	6682.35	150.95	0.04	203.36	134.7%
11:10:07 AM	2.05	109.04	1186.54	0.03	3473.67	0.00	0.00	0.00	0.0%
11:40:07 AM	34.37	322.92	985.11	0.06	7772.82	144.28	0.04	182.32	126.3%
12:10:07 PM	159.95	613.66	6161.26	1.39	11589.10	350.65	1.88	451.17	128.0%
12:40:04 PM	13.61	73.14	377.26	0.16	447.99	0.00	0.00	0.00	0.0%
01:10:07 PM	3.77	74.47	274.64	0.09	318.77	0.00	0.00	0.00	0.0%
01:40:07 PM	73.86	106.87	1182.32	1.40	2374.44	0.00	0.00	0.00	0.0%
02:10:07 PM	66.60	137.36	2098.94	0.72	4756.24	0.00	0.00	0.00	0.0%
Average:	76.30	161.63	3253.97	0.30	7182.09	38.39	0.09	56.67	147.3%

Fig.12 Ubuntu paging and memory information from stat file

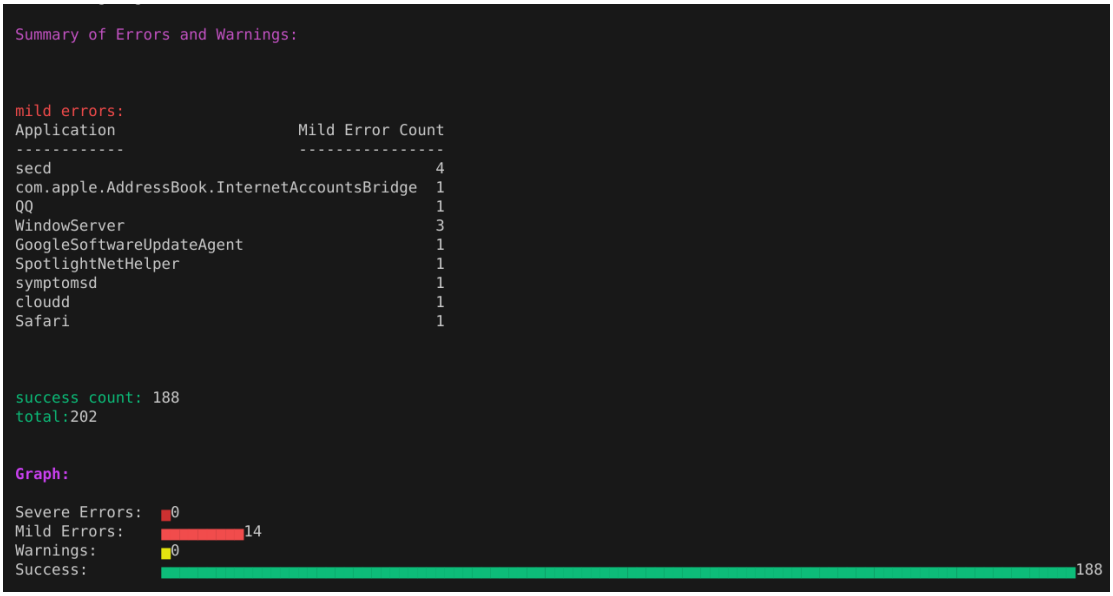


Fig 13 MacOS system log analysis result

```
Error Table:

Error               App Name        Count      Error Message
oom                 N/A             0          oom
segfault            N/A             0          segfault
malloc              N/A             0          malloc
kernel bug          N/A             0          kernel bug
critical            N/A             0          critical
disk failure        N/A             0          disk failure / I/O error
process crash       N/A             0          process crash
process hang        N/A             0          process hang
process killed      N/A             0          process killed
cpu error           N/A             0          cpu fault
gpu error           N/A             0          gpu failure
```

Fig .14 MacOS system log analysis result for severe errors

```
Summary of Errors and Warnings:

severe errors:
Application              Severe Error Count
-----------              ------------------
genunix                  55

mild errors:
Application              Mild Error Count
-----------              ----------------
genunix                  53

warning:
Application              Warning Count
-----------              -------------
genunix                  10


success count: 264
total:382
```

Fig.15 SolarisOS log file analysis

Fig .16 SolarisOS log file analysis graph generation on shell

Error Table:

Error	App Name	Count	Error Message
out of memory	N/A	0	out of memory
oom	N/A	0	oom
segfault	N/A	0	segfault
malloc	N/A	0	malloc
PAM error	N/A	0	PAM_ERROR_MSG
login failure	N/A	0	repeated login failures
account disabled	N/A	0	Account is disabled
connection issue	N/A	0	connection refused or d
cpu error	N/A	0	cpu fault
gpu error	N/A	0	gpu failure

Fig.17 SolarisOS log file analysis result for some severe errors

Action	Occurrences
Security State Change	2
Logon	72
Special Logon	66
Other System Events	13
Logoff	4
User Account Management	2421
System Integrity	5
Security Group Management	1

Fig.18 Windows security log analysis

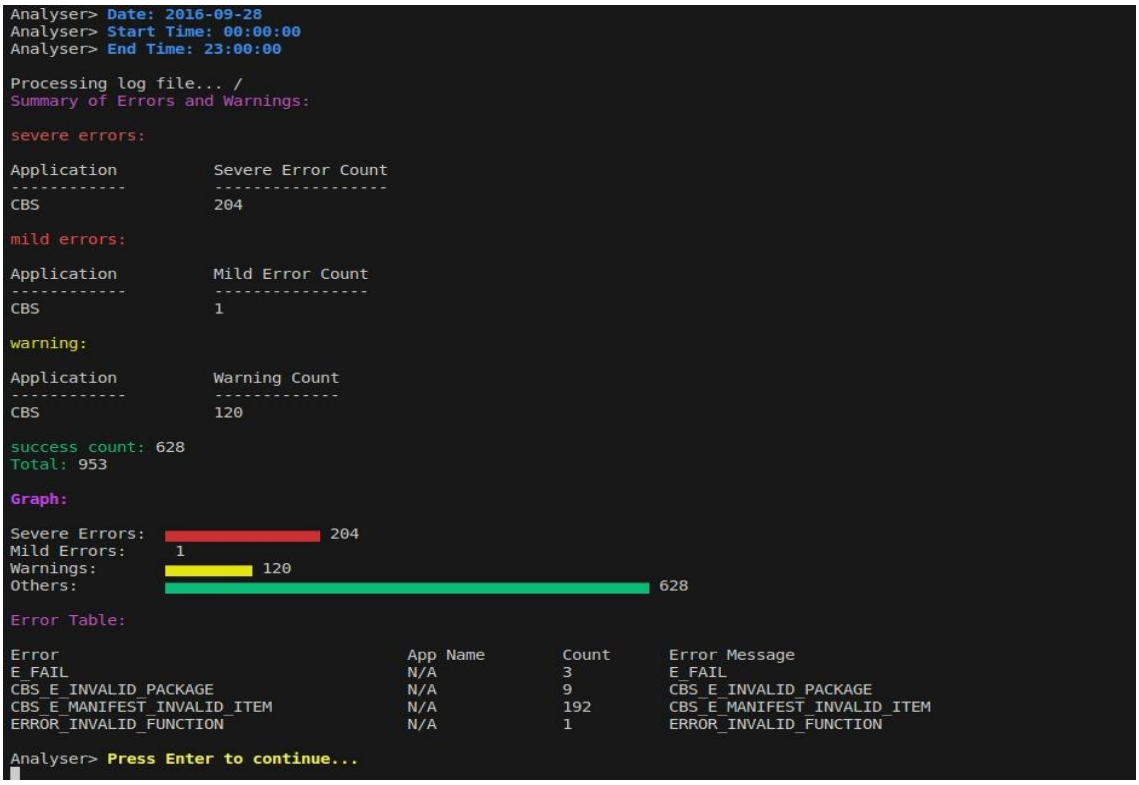


Fig.19 Windows CBS log file analysis

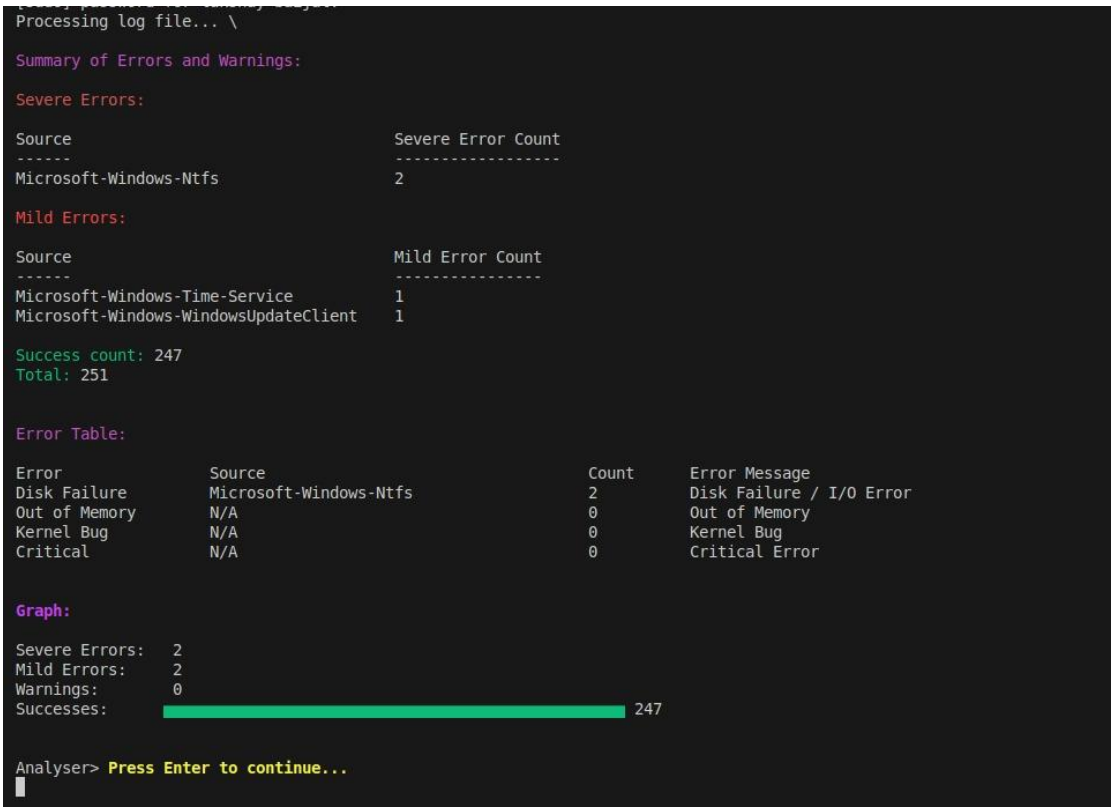


Fig.20 Windows system log analysis result