**Oradio Logging System**

**1. Introduction**

The Oradio project, developed by Oradio Stichting, leverages a comprehensive logging system for USB detection, insertion, and removal services. This logging system is designed to facilitate the monitoring, debugging, and maintenance of the Oradio system by providing detailed and real-time log messages. The logging system incorporates various features, including dynamic log level adjustment, remote monitoring, fault detection, and log file rotation, ensuring robust performance under different conditions.

This document describes the architecture, functionality, and components of the Oradio logging system. The system is built to handle USB detection events and system errors, providing helpful diagnostic information through structured logs.

**2. System Overview**

The logging system in the Oradio project is crucial for debugging and system monitoring. The system captures and logs information about different events, including system crashes, USB device interactions, and warnings. The core features of the system include:

* **Color-Coded Log Output**: Differentiating log levels visually through color.
* **Log Level Management**: Dynamically adjusting log verbosity (e.g., DEBUG, INFO, WARNING, ERROR).
* **Remote Monitoring Integration**: Sending critical log messages to an external service for real-time monitoring.
* **Fault Handling**: Catching and logging low-level crashes.
* **Log File Rotation**: Automatically rotating log files based on size constraints.

Each of these features helps ensure that the system provides comprehensive diagnostics without overwhelming users or system resources.

**3. Key Components of the Logging System**

**3.1. Logging Configuration**

The logging configuration is at the heart of the system. It sets up the overall structure and behavior of the log output. The configuration includes:

* **Log Directory Setup**: Ensures that the log directory exists and is properly configured. If it doesn't exist, it is created automatically using os.makedirs().
* **Loggers**: The primary logger for the system is named oradio\_log. The logger is initialized with the name 'oradio' and set to a default log level of DEBUG. This logger handles all logging output for the Oradio system.
* **Log Handlers**: The script uses multiple handlers to define where the log messages are sent. There are three main handlers:
  + **File Handler**: Logs messages to a file (oradio.log). This handler supports log rotation and ensures that logs do not grow too large by rotating after reaching a size of 512KB and keeping two backups.
  + **Console Handler**: This handler sends log messages to the console (standard error) when the program is run in an interactive terminal.
  + **Remote Monitoring Handler**: Logs WARNING and ERROR level messages and sends them to an external monitoring service for further analysis and tracking.

**3.2. Log Format and Colorization**

To make logs more readable and user-friendly, the system uses a custom log formatter called ColorFormatter. This formatter applies different colors to the log messages based on their severity level. This helps users easily identify and prioritize important log messages.

The ColorFormatter class defines the color codes for different log levels:

* **DEBUG**: Grey
* **INFO**: White
* **WARNING**: Yellow
* **ERROR**: Red

Each log message includes a timestamp, the source filename and line number, the severity level, and the message itself, formatted in a consistent and color-coded manner.

**3.3. Throttled Filter**

The ThrottledFilter class ensures that logging does not occur when the Raspberry Pi is in a throttled state. This is crucial because logging to an SD card during such a state can corrupt the card or cause performance issues.

The ThrottledFilter imports the function get\_throttled\_state\_rpi() from the oradio\_utils module, which checks the system’s throttled state. If the system is throttled, the filter prevents log messages from being written to the log file.

**3.4. Remote Monitoring Integration**

A unique feature of the Oradio logging system is its integration with an external remote monitoring service. This service is particularly useful for monitoring critical events, such as errors or warnings, in real-time.

The RemoteMonitoringHandler is a custom handler that sends error and warning log messages to the monitoring service. This handler is particularly valuable for systems deployed in the field or remotely, where immediate attention may be required for issues like system crashes or hardware failures.

**3.5. Fault Handler**

To ensure that any low-level crashes (e.g., segmentation faults or other system-level errors) are captured, the script utilizes Python’s faulthandler module. This module helps in capturing crashes and printing their stack trace, making it easier to debug critical failures that occur during the execution of the program.

The faulthandler.enable() function activates the fault handler, which provides detailed crash reports to the log.

**3.6. Log Level Management**

The Oradio logging system supports dynamic adjustment of the log level based on user input. The system provides the following log levels:

* **DEBUG**: Logs everything, including debug, info, warning, and error messages.
* **INFO**: Logs informational messages, warnings, and errors.
* **WARNING**: Logs only warnings and errors.
* **ERROR**: Logs only error messages.

This functionality allows users to control the verbosity of the logs based on their needs, such as reducing noise in production environments or enabling detailed debug output during development.

**4. Test Options and User Input Handling**

The script includes a menu-driven interface that allows the user to select various test options. These options include testing different log levels, simulating unhandled exceptions, and generating segmentation faults.

For example:

* **Test Log Levels**: Users can select a log level (DEBUG, INFO, WARNING, ERROR) to test how each level filters the log messages.
* **Unhandled Exception Simulation**: The system can simulate unhandled exceptions in both threads and processes. This is useful for testing how the system reacts to unexpected errors.
* **Segmentation Fault Simulation**: The script can simulate a segmentation fault to test the system's fault tolerance and the effectiveness of the faulthandler module.

The menu-driven system makes it easy for users to test the functionality of the logging system in various scenarios without needing to modify the code directly.

**5. Log File Rotation and Backup**

Log file rotation is implemented using the ConcurrentRotatingFileHandler. This handler allows the log file to grow up to a specified size (512 KB) before being rotated. When the log file reaches its size limit, it is archived, and a new log file is created. The system retains up to two backup files, ensuring that logs are kept manageable and storage space is not wasted.

This feature is crucial for systems with limited storage, such as embedded devices running on SD cards, where log files can quickly consume valuable disk space.

**6. Conclusion**

The Oradio logging system is a critical component of the Oradio project, providing real-time diagnostics and monitoring capabilities. It integrates several advanced features, including color-coded log outputs, dynamic log level management, remote monitoring, fault handling, and log rotation. These features ensure that the system operates efficiently while providing the necessary tools for debugging and maintaining the Oradio service.

With its focus on reliability and usability, the Oradio logging system helps ensure that developers and system administrators can easily monitor system health, detect issues early, and respond to critical failures quickly. The integration of remote monitoring also adds an additional layer of visibility, making it ideal for deployed systems that require constant oversight.

The logging system is designed to be robust, easy to use, and adaptable to different operating environments, making it a valuable asset for any Oradio deployment.