Log Analyzer Project Documentation

Author: Hussain

Date: 22/07/2025

Efficient C++ Log Processing & Analysis Tool

Abstract

This project is a C++ based Log Analyzer tool designed to parse structured log files, aggregate log level statistics by date, and store the results in a SQLite database. It demonstrates effective use of object-oriented programming, memory management, and database integration with SQLite for scalable log data analysis.

Introduction

Log files are essential for monitoring software systems and troubleshooting issues. This project processes log files to extract meaningful statistics, such as counts of different log levels (INFO, ERROR, WARN) over time, which can help developers and system administrators understand system behavior and identify problems.

Objectives

- Develop a modular C++ application for log file parsing.
- Implement date-wise aggregation of log levels.
- Store aggregated data into an SQLite database for persistent storage.
- Demonstrate best practices in OOP, RAII, and memory management.
- Provide a command-line interface to process logs and display statistics.

System Design / Architecture

The system consists of the following main components:
- LogProcessor: Responsible for reading log files, parsing lines, and maintaining statistics.
- SQLiteHandler: Manages SQLite database operations including creating tables and inserting data.
- Main application (main.cpp): Orchestrates the workflow by initializing components, processing logs, and saving results.

Technologies Used

- C++ (Standard Template Library for containers and string handling)
- SQLite (embedded database for storing aggregated statistics)
- Object-Oriented Programming (classes and encapsulation)
- RAII (Resource Acquisition Is Initialization) for safe resource management
- C++11/14 features (smart pointers, auto keyword, etc.)

Implementation Details

Key classes:
- LogProcessor: Implements methods to open and read log files, parse each line to extract date and log level, and aggregate counts in an unordered_map.

- SQLiteHandler: Handles opening/closing the SQLite database, creating tables if they do not exist, and inserting or updating log statistics.

The LogProcessor reads each line, extracts the date and log level, and updates an internal data structure. After processing, statistics are saved to the database using SQLiteHandler.

How to Build and Run

1. Ensure SQLite development files (headers and libraries) are available and linked.
2. Build the project using your preferred C++ compiler or IDE (Visual Studio setup was demonstrated).
3. Place your log files in the 'logs/' directory.
4. Run the executable; it will process the logs, print statistics to the console, and save results in 'log_stats.db'.

Sample Input/Output

Sample log line:
[2025-07-20 10:15:45] INFO Starting process...

Sample output:
=== Log Level Statistics by Date ===
2025-07-20:
 INFO: 2
 ERROR: 2
 WARN: 1

Challenges & Solutions

- Parsing structured log lines with varying formats was handled using careful string operations.
- Managing SQLite integration required ensuring correct database connection handling and SQL statement preparation.
- Implementing RAII principles helped in automatic resource cleanup and avoiding leaks.
- Ensuring clear modularity made the code maintainable and extensible.

Future Work

- Add multi-threaded log processing for improved performance on large datasets.
- Implement a custom memory pool allocator for optimized memory management.
- Extend support for different log formats.

- Create a GUI or web interface for easier interaction.

References

- SQLite official documentation: https://sqlite.org/docs.html
- C++ STL documentation: https://en.cppreference.com/w/
- RAII and modern C++ best practices