Monitoring System Documentation

Teem: 'Retarded children of the 2000s'

May 12, 2023

Contents

Abstract

The Monitoring System is a software project aimed at implementing a program for monitoring the main indicators of a system. It consists of a kernel and multiple agents that collect metrics and send them to the kernel. The kernel logs the metrics and sends notifications to the user when critical values are reached. This documentation provides an overview of the project and guides developers on how to implement and use the Monitoring System.

1 Project Overview

The Monitoring System project involves the following components:

- `Kernel` : The central component responsible for collecting metrics, logging them, and sending notifications.
- `Agents` : Lightweight programs that collect specific metrics and pass them to the kernel.

The project follows the architecture commonly used in existing monitoring systems like Zabbix, Grafana, and Nagios. It employs a kernel-agent model, where the kernel collects metrics and agents collect specific metrics. The agents are dynamically connected and disconnected, allowing flexibility in the system.

2 System Architecture

The Monitoring System architecture is based on the following principles:

- `Kernel` : Collects metrics and logs them.
- `Agents` : Collect specific metrics and send them to the kernel.
- `Metrics` : Indicators to monitor, such as CPU load, memory usage, network throughput, etc.
- `Log File` : Stores the actual metric values periodically.
- `Configuration` : Allows customization of agent settings and critical metric values.
- `User Interface` : Provides a user-friendly interface for displaying log entries and managing agents.

3 Implementation Details

3.1 Programming Language and Code Structure

The Monitoring System is implemented in C++ using the C++17 standard. The code is organized in the following structure:

- `src` folder: Contains the program code.
- `Google Style Guide` : Code style conventions based on the Google C++ Style Guide must be followed.

3.2 Building and Installation

The program must be built using a Makefile. The Makefile should include the following targets:

- `all` : Builds the program.
- `install` : Installs the program in a user-defined directory.
- `uninstall` : Removes the installed program.
- `clean` : Cleans up generated files.
- `dvi` : Generates documentation in DVI format.
- `dist` : Creates a distribution package.
- `tests` : Runs unit tests using the GTest library.

3.3 Model-View-Controller (MVC) Pattern

The Monitoring System follows the MVC pattern to separate business logic from view and controller components. The key principles of MVC implementation in the Monitoring System are:

- `View` : Contains the interface code and is responsible for displaying information to the user. It should not contain any business logic.
- `Controller` : Handles user interactions and acts as a mediator between the view and the model.
- `Model` : Contains the core business logic of the application, including the kernel, agents, and metric processing.

3.4 Logging and Metrics

Instead of using a database, the Monitoring System utilizes a single log file to store the metrics. The log file is organized as a list of actual metric values written periodically at the specified time intervals. Each log entry has the following format:

```
[25.01.2023] | Metric1:  Value1 | Metric2:  Value2 | Metric3:  Value3 | ...
```

- `<TIMESTAMP>` : Timestamp in the format yy-M-M-dd HH:mm:ss.
- `<MetricN>` : Name of the Nth metric.
- `<ValueN>` : Value of the Nth metric.

3.5 Agents and Dynamic Libraries

Agents are implemented as dynamic libraries (‘*.so‘) that can be dynamically connected and disconnected from the kernel while it is running. Each agent should have an ‘updateMetrics()‘ method to load and provide actual metric values. Agents run in the background, collecting metrics at a time interval specified in the configuration file.

3.6 Configuration File

The configuration file allows customization of agent settings and critical metric values. It provides the ability to change the following:

- `Agent name`
- `Agent type`
- `List of critical metric values`
- `Metrics update time`

3.7 User Interface

The Monitoring System's user interface should provide the following features:

- Display the last 20 lines of the log, dynamically updating with each new log entry.
- Show a list of connected agents.
- Detailed information about each agent, including type, monitored metrics, time elapsed since start, and metrics update time.
- Dynamically change the configuration of running agents and save the changes to the configuration file.
- Disconnect selected agents from the list.

3.8 Notifications

To send notifications about critical situations, a Telegram bot is developed. The bot sends messages when metrics reach critical values, including the computer name, metric name, and its value. Users can enable/disable duplication of notifications to a specified email address through the interface. External C++ libraries for Telegram and email communication can be used to implement the notification logic.

3.9 Additional Agent and Metrics

The Monitoring System includes at least one additional agent that collects extra metrics. The additional agent should collect the following metrics:

- `[double]` CPU load by privilege level : idle, user, privileged, dpc, interrupt (percentage for each level) (`cpu_idle_usage, cpu_user_usage, ...`)
- `[double]` Total swap volume (`total_swap`)
- `[double]` Amount of swap used (`used_swap`)
- `[int]` Number of processes ready to run in the queue (refer to process states in Unix if necessary) (`proc_queue_length`)
- `[double]` Counting full and free virtual memory (`virtual_mem_volume, virtual_mem_free`)
- `[int]` Total number of inodes (`inodes`)https://www.overleaf.com/project/62adc83f2b843ff9d3ac2
- `[double]` Average hard disk read time (`hard_read_time`)
- `[int]` Number of errors from the system log (`system_errors`)
- `[int]` Number of user authorizations (`user_auths`)

These additional metrics expand the monitoring capabilities of the system and provide more detailed information about the system's performance and resource utilization.

Conclusion

This documentation provides an overview of the Monitoring System project, its architecture, implementation details, and features. It serves as a guide for developers to understand and implement the system based on the provided technical requirements.

By following the guidelines and utilizing the specified technologies, such as C++17, Makefile, Google Style Guide, and MVC pattern, developers can build a robust and efficient monitoring system capable of collecting and logging various metrics, displaying them in a user-friendly interface, and sending notifications when critical values are reached.

The additional agent and metrics further enhance the system's monitoring capabilities, providing a comprehensive view of the system's performance and resource utilization.

With this documentation as a reference, developers can proceed with the implementation of the Monitoring System project. It is recommended to follow the outlined structure, adhere to the specified programming language and code style guidelines, and make use of the suggested libraries and technologies.

Throughout the development process, it's important to focus on modularization, unit testing, and ensuring the separation of concerns between the kernel, agents, view, and controller components. Additionally, thorough testing and error handling should be implemented to ensure the system functions reliably and effectively.

Remember to regularly refer back to this documentation for guidance and clarification. Good luck with the implementation of the Monitoring System project!