## Table of Contents

## Introduction

The purpose of this Micropython code is to implement a robust event logging system using a dual buffer design. It provides an efficient mechanism for recording events with attributes such as event_id, event type, timestamp and data. The system ensures data integrity by alternating between two memory buffers, and periodically flushing events to persistent storage to minimize data loss during unexpected resets.

The code is designed for environments running Micropython typically used in embedded systems and Iot devices. It is suitable for platforms such as ESP32, Raspberry Pi and similar Micropython compatible devices. This detailed implementation addresses the need for lightweight and reliable event logging in constrained resource environments.

## Overview

1. **Dual buffer:** We use 2 main buffers (bufferA and buffer) and toggling between them for writing and logging operations.
2. **Event Structure:** An event is represented by dictionary containing event_id, timestamp and data.
3. **Flushing mechanism:** Whenever buffer becomes full or the periodic timeout reaches, then we flush it to persistent storage.
4. **Buffer swapping:** We switch the buffers after each flush.
5. **Timer:** We use inbuilt library of Micropython (Timer) for triggering periodic flushing.
6. **Boot Initialization:** Initialize buffers and start the periodic timer.

## Implementation

1. ### Key requirements from the paper for data preprocessing

   Dynamic Logging: Ability to log multiple event types and attributes as per event format provided by the application developer.

   Data Cleaning layer: Parse and clean raw data according to predefined formats.

   Data Labelling layer: Separate “good” and “bad” events based on user-defined predicates.

2. ### Class Initialization

   Logger(buffer_size=100, flush_interval=5), it creates an instance of the Logger class with specified buffer size and flush interval.

   Parameters:

   - buffer_size(type int): Maximum number of events a buffer can hold before it is flushed to storage. Defaults to 100.
   - flush_interval(type int): The time interval(in seconds) at which the logger automatically flushes the buffer to storage. Defaults to 5.

3. ### Functions

   1. #### start_timer()

      This is for the periodic flush timer. This method gets called internally during initialization and after every flush operation.

   2. #### log_event(event_id, data)

      To log an event to the active buffer. If the active buffer exceeds the specified buffer_size, it will trigger an immediate flush.

      Parameters:

      - event_id(type int): A unique identifier for the type of event.
      - data(type dictionary): This is for the additional metadata or information about the event.

      Event Structure: Each event is logged as a dictionary with the below keys.

      - eventid: unique identifier for the event.
      - timestamp: unix timestamp(in seconds) when the event was logged.
      - data: associated data provided during the logging.
        logger.log_event(1, {“key”: “value1”})

   3. #### flush_to_storage()

      Flushes the active buffer to storage, swaps it with the inactive buffer and restarts the flush timer.

      Functionality:

      - Acquires a lock to ensure thread safety.

- Swaps the active buffer with the inactive buffer.
- Writes the inactive buffer contents to storage using write_to_flush().
- Clears the inactive buffer.
- Restarts the flush timer.

4. write_to_flash(buffer)

   This simulates writing the contents of a buffer to storage.

   Parameters:

   - buffer(type list): buffer to be written to storage.

   Functionality: It prints the number of events being flushed and their details.

5. stop()

   stops the logger and flushes any remaining data in the buffers to storage.

   Functionality:

   - Cancels the periodic flush timer.
   - Flushes both buffers to ensure no data is lost.

## Installation

1. Install an IDE like Visual Studio Code with Pymakr plugin.
2. Install the drivers
   i. Install the Silicon Labs CP210x drivers if not installed already.
3. Set up python on your system. And install rshell for file management. This is optional but it is for better usage.
   a. pip install rshell
4. Connect the device via USB to the system.
5. Open the Pycom Firmware Update tool.
6. Select your device and then confirm if the latest firmware is installed.
7. After connecting, ensure the device is detected by the system.
8. Configure Pymakr plugin
   a. Go to pymakr settings and set the COM port. (COMX on windows)
   b. Set the baud rate to 115200.

## Burning the code

1. Save the script. For eg: main.py
2. Now use pymakr plugin or rshell to upload the file to the device.
   a. For rshell
      i. Rshell –port /dev/ttyUSB0
      ii. Cp main.py /pyboard
   b. Via Pymakr, just click the upload button.
3. Using ls /pyboarad command we can check if the file transfer was successful. Ensure main.py is being listed there.
4. For running the code in the device, run the below command in rshell
   a. rshell –port /dev/ttyUSB0
   b. repl (Read evaluate print loop)
5. It will automatically run the main.py

6. While the code is being running on the device, we can use the shell console to view the output logs.

## Expected Results

```
Stored Event: {'eventId': 1, 'timestamp': 1701111111.111, 'data': {'key': 'value1'}}
```