

## РАЗРАБОТКА ЖУРНАЛА СОБЫТИЙ С ИСПОЛЬЗОВАНИЕМ СТАНДАРТНОЙ БИБЛИОТЕКИ ЯЗЫКА C++

Н.С. Румянцев<sup>1</sup>, С.В. Востокин<sup>1</sup>

<sup>1</sup>Самарский университет, г. Самара  
e-mail: rumancevn07@gmail.com, easts@mail.ru

### Аннотация

Некоторые приложения работают с небольшим объемом хранимых данных, которые можно разместить непосредственно в оперативной памяти. Универсальным способом реализации базы данных в памяти является метод журнала упреждающей записи (WAL). Однако такая реализация требует оптимизации использования памяти. В работе предложен компактный оптимизированный вариант реализации журнала событий в памяти процесса с использованием стандартной библиотеки языка C++ с возможностью дальнейшей модификации.

**Ключевые слова:** журнал событий, C++, оптимизация памяти, операции чтения и записи, ёмкость вектора, размер вектора.

### Введение

Вычисления с журналом событий реализуются через паттерн Write-Ahead Log (WAL) с предварительной фиксацией изменений. В SQL-базах данных WAL обеспечивает сохранение результатов операций INSERT, UPDATE, DELETE, COMMIT и других, включая возможность их отмены [1].

Однако журналирование можно использовать не только в базах данных SQL, но и непосредственно в прикладном коде. Поэтому может возникнуть необходимость в реализации паттерна WAL с помощью промышленных языков программирования, например C++.

Тем не менее, реализация такого решения является сложной технической задачей. При переносе паттерна проектирования между языками программирования необходимо предотвращать появление новых ошибок и минимизировать перенос существующих проблем. Например, одной из проблем WAL является то, что накладные расходы на журналирование могут снизить производительность системы, особенно в случае записи данных на внешние носители в ходе работы. Это может существенным образом повлиять на процесс разработки.

### 1. Общие положения для работы, особенности реализации

Класс журнала – `write_ahead_log`. Журнал в данном контексте представляет собой массив записей об изменении состояний. В журнале не требуется хранить полное состояние, необходимо фиксировать начальное состояние и историю изменений. Необходимо реализовать операции записи (`write`) и чтения (`read`) по индексу. Для моделирования работы журнала и взаимодействующих с ним процессов создается массив произвольных элементов.

При реализации журнала событий на C++ возникает ограничение конечного размера журнала, что может привести к переполнению. В отличие от традиционного файлового хранения, рассматриваемый журнал полностью находится в памяти для упрощения процесса отладки.

## 2. Работа с журналом, размер и ёмкость журнала

Журнал событий может быть неограниченным, ограниченным и настраиваемым. В качестве примера рассматривается ситуация с настраиваемым журналом, являющаяся самой неопределенной, где на одну запись (write) в журнале приходится от 1 до N чтений (read). Журнал событий в коде представляет собой вектор, который заполняется во время функционирования кода. Это означает, что при работе с журналом применяются методы, предусмотренные для векторов языка C++. Это является основным фактором, позволяющим решить проблему с переполнением журнала.

В C++ вектор характеризуется размером (size) и ёмкостью (capacity). Размер вектора, возвращаемый функцией size(), определяет количество элементов, содержащихся в контейнере. Ёмкость вектора, возвращаемая функцией capacity(), определяется как количество элементов, для которых вектору выделена память, без выделения дополнительной памяти. Ёмкость вектора всегда не меньше его размера, то есть выполняется условие:  $capacity \geq size$ . Изменение ёмкости вектора носит скачкообразный характер. Ёмкость и размер должны масштабироваться пропорционально количеству записей во время выполнения, поэтому функции size() и capacity() интегрируются непосредственно в метод write [2].

Таблица результатов исследования влияния параметра NUM\_THREADS на работу журнала событий представлена ниже.

Таблица 1 – Результаты исследования числовых характеристик

| NUM_THREADS | Size | Capacity | NUM_THREADS | Size | Capacity |
|-------------|------|----------|-------------|------|----------|
| 15          | 45   | 63       | 5           | 15   | 19       |
| 14          | 42   | 42       | 4           | 12   | 13       |
| 10          | 30   | 42       | 3           | 9    | 9        |
| 9           | 27   | 28       | 2           | 6    | 6        |
| 7           | 21   | 28       | 1           | 3    | 3        |
| 6           | 18   | 19       | 0           | 0    | 0        |

Зависимость размера журнала (SIZE) и ёмкости журнала (CAPACITY) от количества потоков (NUM\_THREADS) можно установить на основании эмпирических данных. Размер журнала вычисляется по формуле  $SIZE = 3 * NUM\_THREADS$ . Ёмкость журнала вычисляется по формуле(1):

$$a_n = [a_{n-1} * 1,5], \quad (1)$$

где  $a_n$  – текущий член ряда, а  $a_{n-1}$  – предыдущий член ряда. Квадратные скобки обозначают антье – целую часть действительного числа.

Анализ затрат памяти на работу журнала событий показал, что оптимизация памяти, выражаемая разницей ёмкости и размерности, является достижимой целью. Для этого в C++ существует специальный метод shrink\_to\_fit(), уменьшающий ёмкость вектора до его размера. Принимая во внимание всё описанное выше, код журнала представлен на рисунке 1 ниже.

```

class write_ahead_log {
public:
    virtual void write(unsigned& index, unsigned tag, const std::string& blob) {
        setlocale(LC_ALL, "Russian");
        std::unique_lock<std::mutex> lock(mut);
        log.push_back(std::pair<unsigned, std::string>(tag, blob));
        index = log.size() - 1;
        log.shrink_to_fit();
        //std::cout << "ЗАПИСЬ: " << log.size() << std::endl;
        //std::cout << "ЕМКОСТЬ: " << log.capacity() << std::endl;
    }
    virtual bool read(unsigned index, unsigned& tag, std::string& blob) {
        setlocale(LC_ALL, "Russian");
        //std::cout << "ЕМКОСТЬ: " << log.capacity() << std::endl;
        std::unique_lock<std::mutex> lock(mut);
        if (index < log.size()) {
            tag = log[index].first;
            blob = log[index].second;
            return true;
        }
        return false;
    }
private:
    std::vector<std::pair<unsigned, std::string>> log;
    std::mutex mut;
};

```

Рисунок 1 – Код журнала событий

Blob – запись о состоянии, а tag – вспомогательный элемент записи. Класс `unique_lock` обеспечивает эксклюзивное владение мьютексом (`mut`). Это означает, что в любой момент времени только один поток удерживает конкретный мьютекс.

Таким образом, был получен выигрыш памяти, который измеряется от 0 до N от каждой итерации вектора, где N – это разность между ёмкостью и размерностью. Средний выигрыш памяти измеряется по формуле (2) и равен ~16,7%.

$$1 - \left( \frac{\left( \left\lfloor \frac{a_n-1}{3} \right\rfloor * 3 \right) + 3}{a_n} + \frac{a_n - \left( \left\lfloor \frac{a_n}{3} \right\rfloor * 3 \right)}{a_n} \right) . \quad (2)$$

Однако такой способ оптимизации памяти имеет проблемы с производительностью, поскольку функция `shrink_to_fit()` обладает линейной вычислительной сложностью относительно числа элементов в векторе. Прирост эффективности использования памяти, достигаемый с помощью её постоянного перераспределения, подходит далеко не для всех типов систем. Средний показатель деградации производительности определяется как процентное отношение среднего времени выполнения системы в стандартном режиме к среднему времени выполнения системы при использовании метода `shrink_to_fit()` и равен ~6%. Такое уменьшение производительности не является приемлемым для многих систем, требующих высокой производительности.

Тем не менее, возможность реализации подобного механизма перераспределения памяти указывает на потенциал дальнейшей оптимизации использования памяти.

## Заключение

В результате была разработана компактная реализация кода, обеспечивающая логирование, аналогичное механизмам логирования баз данных SQL. Представленный код журнала обладает возможностью модификации, на его основе могут быть собраны различные версии кода, адаптированные под индивидуальные условия программ, требующих логирования. Представленная оптимизация кода является решением проблемы журнала событий с переполнением памяти, благодаря её перераспределению. Среднее улучшение эффективности использования памяти составляет ~16,7%, тогда как средний процент увеличения времени выполнения составляет ~6%. Такое решение подходит не для всех систем, однако является приемлемой модификацией кода, когда увеличение времени приемлемо.

Данная реализация кода журнала соответствует требованиям паттерна программирования WAL и может являться универсальным решением для широкого класса задач, задавая структуру взаимодействия объектов и классов, независимо от финальных вариаций различных программных продуктов [3]. Благодаря простоте реализации, журнал может быть интегрирован в сторонних программных комплексах.

## Список использованных источников

1. Write-Ahead Logging (WAL). PostgreSQL Documentation. – URL: <https://www.postgresql.org/docs/current/wal-intro.html> (дата обращения: 13.03.2026).
2. Липпман, С. Язык программирования C++. Полное руководство / С. Липпман, Ж. Лажойе // Саратов: Профобразование, 2019. – 1104 с.
3. Фримен, Э. Head First. Паттерны проектирования / Э. Фримен, Э. Робсон, К. Сьерра, Б. Бейтс // СПб.: Питер, 2018. – 656 с.

## CROSS-PLATFORM EVENT LOG DEVELOPMENT USING C++

N.S. Rumyantsev<sup>1</sup>, S.V. Vostokin<sup>1</sup>

<sup>1</sup>Samara University, Samara

e-mail: rumancevn07@gmail.com, easts@mail.ru

## Abstract

Some applications work with small amounts of stored data that can be stored directly in RAM. A universal method for implementing an in-memory database is the write-ahead logging (WAL) method. However, this implementation requires memory optimization. This paper proposes a compact, optimized implementation of an in-memory event log using the C++ standard library, with the potential for further modification.

**Keywords:** Event logs, C++, memory optimization, read and write operations, vector capacity, vector size.