ТИТУЛЬНЫЙ ЛИСТ

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc102750548)

[ОСНОВНАЯ ЧАСТЬ 6](#_Toc102750549)

[ЗАКЛЮЧЕНИЕ 7](#_Toc102750550)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 8](#_Toc102750551)

# ВВЕДЕНИЕ

Идея создания многопоточных подходов появилась и использовалась ещё в прошлом веке. Но бурное развитие многопоточных алгоритмов началось только в 2000-ых годах в связи с повсеместным распространением многоядерных процессов. Это позволило писать большое количество эффективного по памяти и структурно оптимального многопоточного кода.

В многопоточных приложениях потоки или отдельные единицы исполнения кода зачастую решают одну задачу. Поэтому им необходима синхронизация через передачу друг другу данных, обмениваться сигналов или сообщений. Потоки могут делать это посредством как *блокирующей синхронизации*, так и *неблокирующей*.

Блокирующая синхронизация использует специальные примитивы синхронизации, чтобы в любой момент только один поток мог исполняться в *критической секции*. Другие потоки переводятся операционной системой в режим сна и ожидают пробуждения, чтобы попытаться исполниться в критической секции в будущем. Такой подход может привести к серьёзным проблемам: дедлок, ухудшение быстродействия из-за затрат на перевод в сон или пробуждение.

Альтернативой служит неблокирующий подход, который не переводит ни один поток в состояния сна, а заставляет беспрерывно исполнять код. В этом случае понятие критической секции, в которой в любой момент времени может находится лишь один поток, уходит. Точнее, теперь блокирующая секция сужается лишь до одной инструкции ассемблерной инструкции *compare-and-swap*. И только этот код может исполняться потоком эксклюзивно.

Данный подход во многих случаях эффективнее по времени и использует возможности современных процессоров на всю мощность.

В свою очередь, неблокирующие алгоритмы могут быть поделены на три типа в зависимости от гарантий, которые они дают:

Таблица 5.1 – Типы неблокирующих алгоритмов и их гарантии

|  |  |
| --- | --- |
| Тип | Гарантии |
| Без препятствий | Самая слабая из гарантий. Поток совершает прогресс, если не встречает препятствий со стороны других потоков. Алгоритм работает без препятствий, если поток, запущенный в любой момент (при условии, что выполнение всех препятствующих потоков приостановлено), завершит свою работу за детерминированное количество шагов. Синхронизация с помощью мьютексов не отвечает даже этому требованию: если поток остановится, захватив мьютекс, то остальные потоки, которым этот мьютекс нужен, будут простаивать. |
| Без блокировок | Для алгоритмов без блокировок гарантируется системный прогресс по крайней мере одного потока. Например, поток, выполняющий операцию «сравнение с обменом» в цикле, теоретически может выполняться бесконечно, но каждая его итерация означает, что какой-то другой поток совершил прогресс, то есть система в целом совершает прогресс. |
| Без ожиданий | Самая строгая гарантия прогресса. Алгоритм работает без ожиданий, если каждая операция выполняется за определённое количество шагов, не зависящее от других потоков. |

Из трёх вышеприведённых типов, чаще всего выбирается тип без блокировок или лок-фри. Он является компромиссом между не всегда возможным типом без ожиданий и типом без препятствий, который даёт слишком слабые гарантии. Именно на основе типа без блокировок создано множество лок-фри структур *данных*, которые позволяют производить с ними операции многопоточно.

Одна из таких лок-фри структур данных является Hash Array Mapped Trie. Впервые Hash Trie была описана Брандисом и Фредкином [2]. Позже, она была модифицирована Бегвилом [2] в Hash Trie Mapped Trie – подтип префиксных деревьев, хранящий пары ключ-значения. Поиск ведётся по хешированному ключу. Каждый узел префиксного дерева хранит ссылки на поддеревья внутри массива, которые индексируется с помощью bitmap. Это делает структуру эффективной по памяти и использованию кэша.

В 2017 году трое учёных [1] предложили многопоточный алгоритм реализации Hash Array Mapped Trie. В данном алгоритме, они избавились от проблемы stop-the-word, которая свойственна хеш-таблицам; а также сделали структуру данных эффективной по памяти – с динамическим c глубины.

В данный момент реализация лок-фри Hash Array Mapped Trie есть на языке Java, но отсутствует для С++. Так, в библиотеке лок-фри структур данных libcds [3] необходима реализация этой структуры данных на языке С++. Именно это и является главной целью данной работы.

В качестве задач, шагов к выполнению были предложены следующие:

1. Анализ Hash Array Mapped Trie структуры данных
2. Реализация однопоточной версии
3. Реализация блокирующей многопоточной версии
4. Переход к неблокирующей многопоточной версии HAMT
5. Решение проблемы ABBA
6. Внедрение в библиотеку libcds
7. Добавление в libcds

# ОСНОВНАЯ ЧАСТЬ

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Aleksandar Prokopec, Phil Bagwell, Martin OderskyCache, Aware Lock-Free Concurrent Hash Tries
2. Fredkin, E. 1960. Trie memory. Communications of the ACM 3
3. Phil Bagwell. Ideal Hash Trees