ТИТУЛЬНЫЙ ЛИСТ

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc102834093)

[ОСНОВНАЯ ЧАСТЬ 6](#_Toc102834094)

[ЗАКЛЮЧЕНИЕ 7](#_Toc102834095)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 8](#_Toc102834096)

# ВВЕДЕНИЕ

Идея создания многопоточных подходов появилась и использовалась ещё в середине прошлого века [4]. Но её бурное развитие началось с повсеместным распространением многоядерных процессов [5]. Это раскрыло потенциал многопоточного кода: использование всей мощности компьютера, а значит высокая производительность.

В многопоточных приложениях потокам (отдельные единицы исполнения кода) зачастую необходима синхронизация через передачу друг другу данных, обмен сигналами, сообщений. Потоки могут делать это посредством как *блокирующей синхронизации*, так и *неблокирующей*.

Блокирующая синхронизация использует специальные примитивы синхронизации (мьютекс, семафор, условные переменные), чтобы в любой момент только один поток мог исполняться в *критической секции*. Для обеспечения эксклюзивности выполнения другие потоки переводятся операционной системой в режим ядра, где они ожидают исполнения. Такой подход может привести к серьёзным проблемам: дедлок, ухудшение быстродействия из-за затрат на перевод режим ядра.

Альтернативой служит неблокирующий подход, который не переводит потоки в режим ядра, а заставляет каждый 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 ассемблерной инструкции *compare-and-swap*. И только этот код может исполняться потоком эксклюзивно. Данный подход во многих случаях эффективнее по времени, так как не требует временных затрат на блокировку.

В свою очередь, неблокирующие алгоритмы могут быть поделены на три типа в зависимости от гарантий, которые они дают [6]:

Таблица 5.1 – Типы неблокирующих алгоритмов и их гарантии

|  |  |
| --- | --- |
| Тип | Гарантии |
| Без препятствий | Любой поток при любой начальной конфигурации завершает свою работу через ограниченное количество шагов. |
| Без блокировок | Хотя бы один из потоков совершает прогресс в любой момент времени. |
| Без ожиданий | Любой поток завершается через конечное количество шагов. И это число не зависит от других потоков |

Из трёх вышеприведённых типов наиболее используемый тип «без блокировок (то есть 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 большое число *лок-фри структур данных*, которые позволяют производить с ними операции многопоточно.

Одной 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, поддерживающие операции поиска значения по ключу, удаление ключа, добавление значения по ключу. Также, операции производятся 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 из Швейцарии [1] предложили многопоточную реализацию *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 ABA
6. Внедрение в библиотеку libcds
7. Добавление в libcds

# ОСНОВНАЯ ЧАСТЬ

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Aleksandar Prokopec, Phil Bagwell, Martin Odersky Cache, Aware Lock-Free Concurrent Hash Tries
2. Fredkin, E. 1960. Trie memory. Communications of the ACM 3
3. History of multithreading [Электронный ресурс] – URL: <https://people.cs.clemson.edu/~mark/multithreading.html#:~:text=History%20of%20Multithreading&text=Summary%3A%20Multithreading%20first%20appeared%20in,investigated%20by%20IBM%20in%201968.&text=Most%20attempts%20at%20a%20history,PPUs%20in%20the%20CDC%206600> (дата обращения 15.04.2022)
4. Multi-core processors – An overview [Электронный ресурс] – URL: <https://arxiv.org/ftp/arxiv/papers/1110/1110.3535.pdf> (дата обращения 15.04.2022)
5. Phil Bagwell. Ideal Hash Trees
6. Yale-WIKI [Электронный ресурс] – URL: <https://www.cs.yale.edu/homes/aspnes/pinewiki/JayantiTanToueg.html> (дата обращения 15.04.2022)