Fine-Grained Locking

Fine-Grained Locking

Задача: реализовать контейнер для работы в многопоточном окружении.

Fine-Grained Locking

Задача: реализовать контейнер для работы в многопоточном окружении.

Coarse-Grained Locking:

Навесить на контейнер один мьютекс, который будет упорядочивать все операции.

+ Простота
– Никакой параллельности

Fine-Grained Locking

Задача: реализовать контейнер для работы в многопоточном окружении.

Coarse-Grained Locking:

Навесить на контейнер один мьютекс, который будет упорядочивать все операции.

+ Простота
– Никакой параллельности

Fine-Grained Locking:

Защитить разные части контейнера разными мьютексами.

+ Параллельность
– Сложность

Хэш-таблица

Хэш-таблица – структура данных, которая реализует операции над множеством элементов:

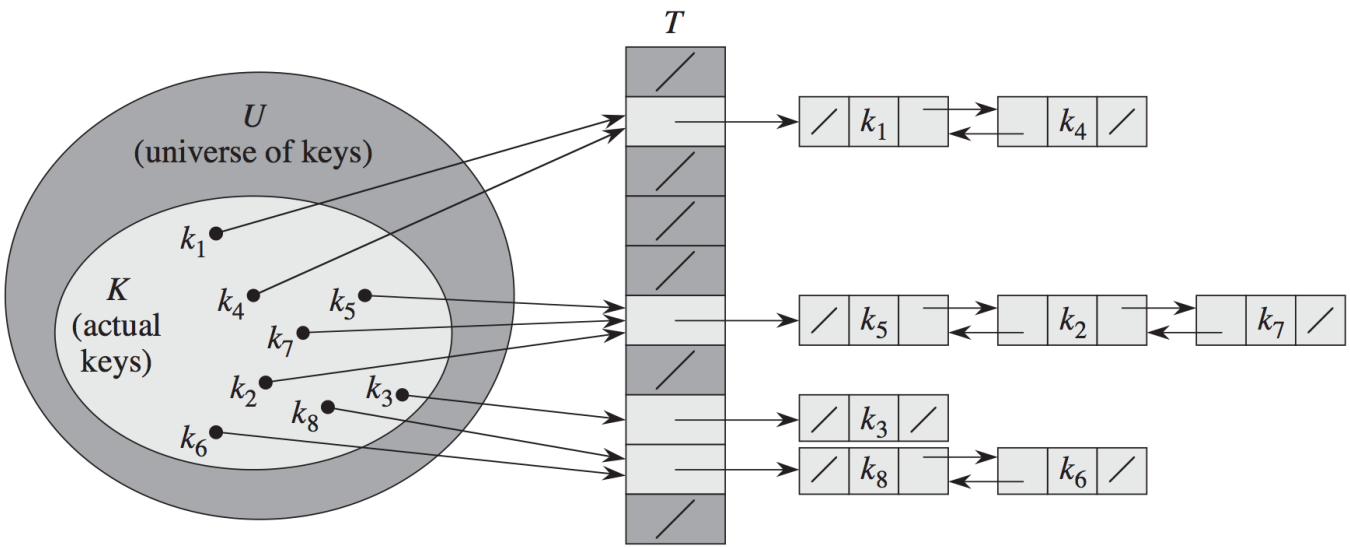
- `insert(x)` – добавить элемент в хэш-таблицу
- `contains(x)` – проверить наличие элемента в хэш-таблице
- `remove(x)` – удалить элемент из хэш-таблицы

Хэширование цепочками

Хэш-функция $h : U \to \{0, \ldots, M-1\}$, где U – множество всех возможных элементов.

Элементы распределяются между m **корзинами** с помощью хэш-функции: элемент x при вставке попадает в корзину с номером $h(x) \bmod m$

Элементы внутри корзины организованы в список.



Хэширование цепочками

Перед выполнением операции нужно вычислить корзину, в которую попал элемент:

$$x \to h(x) \bmod m$$

Операции:

- `insert(x)` – вставка элемента в список корзины
- `contains(x)` – поиск по списку корзины
- `remove(x)` – удаление элемента из списка корзины

Во время вставки и удаления нужно сканировать весь список.

Время работы всех операций линейно зависит от размера корзины.

Расширение таблицы

По мере увеличения числа элементов в таблице списки в корзинах начнут расти.

Время работы всех операций линейно зависит от числа элементов в корзинах, так что списки нужно поддерживать короткими.

Ключевая метрика: **load factor** $= \dfrac{\text{число элементов}}{\text{число корзин}}$

Критерий расширения: load factor $\geqslant L$.

Расширение таблицы:

Увеличение числа корзин + повторная вставка всех элементов

Таблицу расширяет вставка, которая превысила порог.

Fine-Grained Locking

Параллельность заложена в самой идее хэширования цепочками:

> Если потоки работают с разными корзинами, то они могут выполнять свои операции параллельно.

Зафиксируем число мьютексов, оно не будет увеличиваться при расширении таблицы.

Lock Striping

Каждую из корзин хэш-таблицы будет охранять один из мьютексов.

Набор корзин, которые охраняет один и тот же мьютекс, будем называть **страйпом** (*stripe* – полоска).

В пределах одного страйпа потоки работают последовательно, операции над корзинами из разных страйпов могут выполняться параллельно.

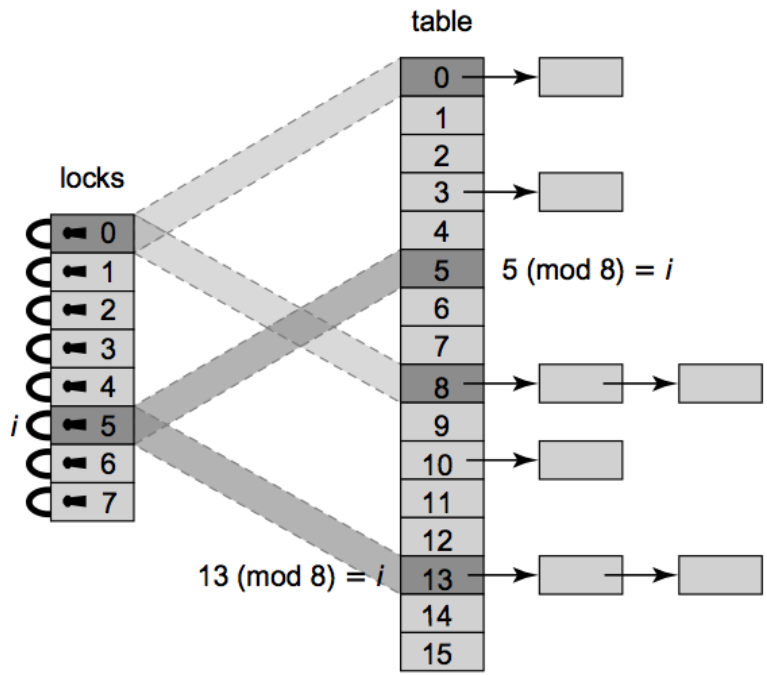
Для выполнения вставки/поиска/удаления поток должен захватить мьютекс страйпа той корзины, в которую попал (мог попасть) элемент.

Для расширения таблицы поток должен получить к ней монопольный доступ, т.е. захватить мьютексы всех страйпов.

Нарезка на страйпы

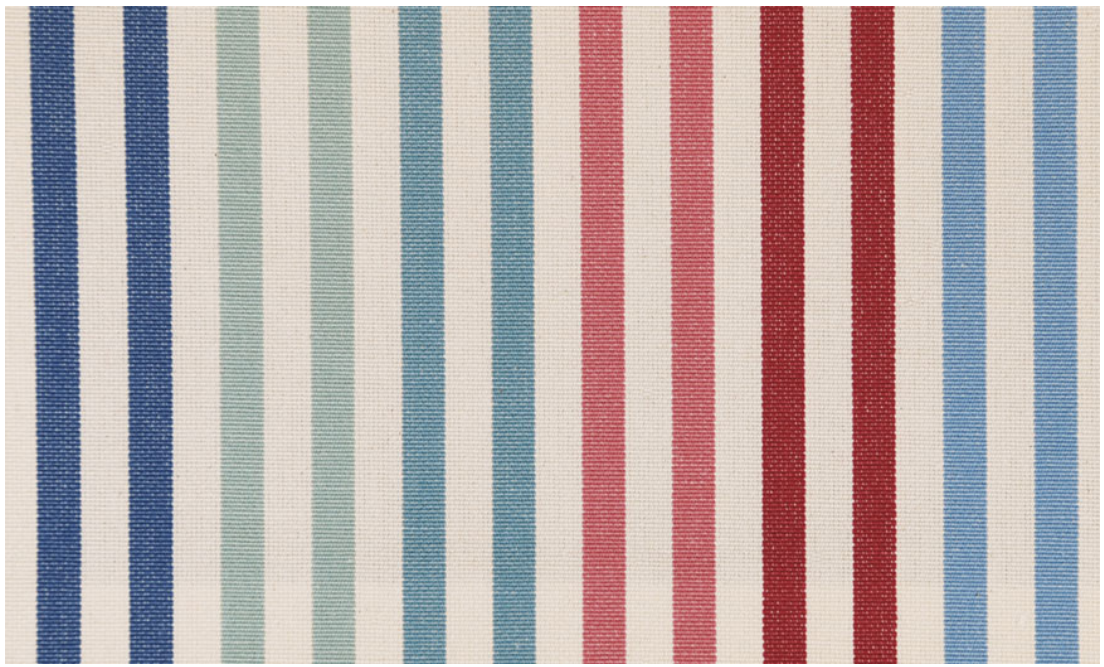
Используем фиксированное число мьютексов!

Корзину k будет защищать мьютекс $k \bmod n$, где n – число мьютексов.



Где же здесь полоски?

Корзину k будет защищать мьютекс $k \bmod n$, где n – число мьютексов.



Трудности

Хэш-таблица должна расширяться по мере заполнения.

В многопоточном мире это может привести к интересным эффектам:

Поток может заблокироваться на захвате мьютекса страйпа, пока в таблице было одно число корзин, а когда поток получит доступ, то она может быть уже другого размера.

При этом номер мьютекса: $\big(h(x) \bmod m\big) \bmod n$.

> До захвата мьютекса мы не знаем число корзин, а значит и корзину, в которой должен быть элемент.

Но как тогда узнать, какой мьютекс захватывать?

Трудности

При расширении таблицы элементы могут менять свои корзины.

А вдруг при этом элемент поменяет мьютекс?

Нужно гарантировать, что при расширении таблицы элементы не перемещаются между страйпами.

К счастью, это возможно!

Сохраняем страйп при расширении

Перед расширением элемент x попадал в корзину $h(x) \bmod m$, после расширения – $h(x) \bmod r \cdot m$, где r – коэффициент расширения.

Номер страйпа/мьютекса для k-ой корзины: $k \bmod n$.

Хотим, чтобы мьютексы до/после расширения не менялись:

$$(h(x) \bmod m) \bmod n = (h(x) \bmod r \cdot m) \bmod n$$

Простое свойство:

$$(a \bmod k \cdot n) \bmod n = a \bmod n$$

.

Идеи?

Сохраняем страйп при расширении

Перед расширением элемент x попадал в корзину $h(x) \bmod m$, после расширения – $h(x) \bmod r \cdot m$, где r – коэффициент расширения.

Номер страйпа/мьютекса для k-ой корзины: $k \bmod n$.

Число корзин должно быть кратно числу мьютексов!

Тогда номер мьютекса, который защищает корзину, не будет зависеть от числа корзин:

Номер страйпа/мьютекса: $h(x) \bmod n$

Как расширить таблицу

Поток захватил мьютекс одного из страйпов, сделал вставку и увидел, что таблицу нужно расширить.

Для того, чтобы выполнить расширение, потоку нужен монопольный доступ ко всем страйпам, т.е. поток должен захватить все мьютексы хэш-таблицы.

Нужно застраховаться от дедлоков:

> **Рецепт**: Дедлоки невозможны, если все потоки захватывают мьютексы в одном и том же порядке.

Как захватить мьютексы всех страйпов?

Как расширить таблицу

Поток захватил мьютекс одного из страйпов, сделал вставку и увидел, что таблицу нужно расширить.

Для того, чтобы выполнить расширение, потоку нужен монопольный доступ ко всем страйпам, т.е. поток должен захватить все мьютексы хэш-таблицы.

Нужно застраховаться от дедлоков:

> **Рецепт**: Дедлоки невозможны, если все потоки захватывают мьютексы в одном и том же порядке.

Как захватить мьютексы всех страйпов?

> Нужно отпустить мьютекс текущего страйпа и захватить все мьютексы по очереди.

Как расширить таблицу

Есть ли в алгоритме гонка?

Как расширить таблицу

Есть ли в алгоритме гонка?

Да!

Пусть два потока в разных страйпах параллельно обнаружили переполнение таблицы.

Каждый из них отпустит мьютекс текущего страйпа, захватит все мьютексы таблицы (сначала один поток, потом другой) и выполнит расширение.

В результате таблица будет расширена дважды!

Как расширить таблицу

Как поток может обнаружить, что другой поток опередил его с расширением?

Как расширить таблицу

Как поток может обнаружить, что другой поток опередил его с расширением?

> Перед тем, как отпускать мьютекс текущего страйпа, поток может запомнить текущее число корзин в хэш-таблице.

В момент, пока он владеет мьютексом своего страйпа, это число заведомо не изменяется.

После того, как поток отпустил мьютекс текущего страйпа и по очереди захватил все мьютексы таблицы, он сравнивает значение, которое запомнил, с актуальным числом корзин:

Если корзин увеличилось, то другой поток его опередил, и расширение делать не нужно.

Как расширить таблицу

Нужно ли потоку захватывать все мьютексы, чтобы понять, что другой поток опередил его с расширением?

Как расширить таблицу

Нужно ли потоку захватывать все мьютексы, чтобы понять, что другой поток опередил его с расширением?

Нет, не нужно!

> Достаточно захватить первый мьютекс и проверить, изменилось ли число корзин в хэш-таблице.

Если первый мьютекс захвачен, то с этого момента никакой другой поток не может выполнить расширение.

Если поток увидел ожидаемое число корзин, то оно не изменится до тех пор, пока поток не захватит оставшиеся мьютексы.

Читатели и писатели

Наблюдение:

Потоки, которые выполняют поиск в пределах одного страйпа, могут работать параллельно, поскольку они не изменяют состояния хэш-таблицы.

Потоки, которые вставляют и удаляют элементы, требуют эксклюзивного доступа к страйпу.

Потоки, которые выполняют поиск, назовем **читателями**, а потоки, которые запускают вставки и удаления – **писателями**.

Задача: придумать механизм синхронизации читателей и писателей для увеличения параллельности.

Бонус: Skip List