Национальный исследовательский университет
компьютерных технологий, механики и оптики

Факультет ПИиКТ

Низкоуровневое программирование
Лабораторная работа №1
Вариант 1

Работу выполнил: Зубахин Д. С.
Группа: P33312
Преподаватель: Кореньков Ю. Д.

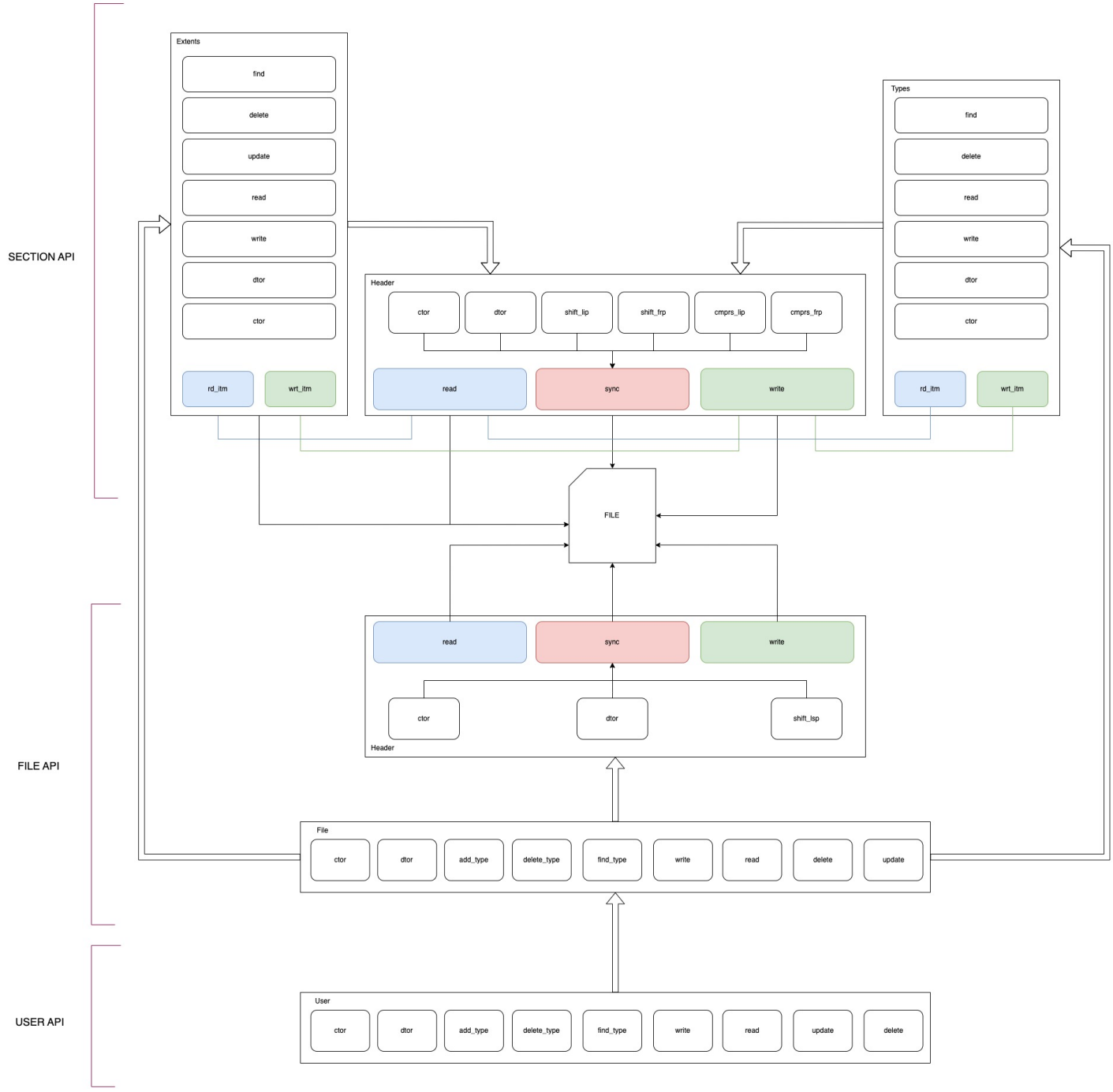
Санкт-Петербург
2022 год

1 Задача

Основная цель лабораторной работы - Создать модуль, реализующий хранение в одном файле данных (выборку, размещение и гранулярное обновление) информации общим объёмом от 10GB соответствующего варианту вида:

1. Спроектировать структуры данных для представления информации в оперативной памяти
2. Спроектировать физическую организацию данных и архитектуру приложения
3. Спроектировать прикладные интерфейсы модулей
4. Сделать имплементацию интерфейсов посредством TDD
5. Покрыть реализацию модульными тестами
6. Написать стресс тесты для приложения на полученном прикладном интерфейсе

2 Описание работы



Данные хранятся как деревья узлов. Каждый узел содержит ключ свой и значение которое является одним из 4 примитовов. Для объектных типов узлов существуют также схемы. Схемы можно добавить, удалить и прочитать с помощью user api. Если мы проверяем ноду на соответствие типу, то все поля в типе являются required. Остальные поля в типе могут быть или не быть.

Также у нас есть запросы. Запросы состоят из частичек запросов, которые содержат ключ, тип и значение ключа. Таким образом запросы могут быть составными.

2.1 Section API

2.1.1 Header API

```
struct Sect_head *sect_head_new();

Status sect_head_ctor(struct Sect_head *const, const Fileoff, FILE *const);
Status sect_head_dtor(struct Sect_head *);

Status sect_head_sync(struct Sect_head *const);

Status sect_head_shift_lip(struct Sect_head *const, const size_t);
Status sect_head_shift_frp(struct Sect_head *const, const size_t);

Status sect_head_cmprs_lip(struct Sect_head *const, const size_t);
Status sect_head_cmprs_frp(struct Sect_head *const);

Status sect_head_read(struct Sect_head *const header, const Sectoff soff,
const size_t sz, void *const o_data);
Status sect_head_write(struct Sect_head *const, const Sectoff, const size_t, const void *const);

Fileoff sect_head_get_fileoff(const struct Sect_head *const, const Sectoff);
Sectoff sect_head_get_sectoff(const struct Sect_head *const, const Fileoff);
```

2.1.2 Extents API

```
struct Sect_ext *sect_ext_new();

Status sect_ext_ctor(struct Sect_ext *const, const Fileoff, FILE *const);
Status sect_ext_dtor(struct Sect_ext *);

Status sect_ext_write(struct Sect_ext *const section, const String *const key,
const void *const data, const size_t data_sz, Entity *const entity, Sectoff *const save_addr);

Status sect_ext_read(const struct Sect_ext *const section, const Sectoff entity_addr,
Entity *const o_entity, Json *const o_json);

Status sect_ext_update(struct Sect_ext *const section, const Sectoff soff,
const String *const new_key, const void *const val, const size_t val_sz, Entity *const entity);

Status sect_ext_delete(struct Sect_ext *const section,
const Sectoff sectoff, Entity *del_entity);

Status sect_ext_load(const struct Sect_ext *const section,
List_Fileoff_itm *const collection);
```

2.1.3 Types API

```
struct Sect_types *sect_types_new();

Status sect_types_ctor(struct Sect_types *const, const Fileoff, FILE *const);
Status sect_types_dtor(struct Sect_types *);

Status sect_types_write(struct Sect_types *const section, const Type *const type, Sectoff *const o_wr
Status sect_types_delete(struct Sect_types *const section, const Sectoff del_soff);
Status sect_types_read(struct Sect_types *const section, Sectoff sctoff, Type *const o_type, Type_ent

Status sect_types_find(struct Sect_types *const section, const String *const type_name, Type *const o
```

2.2 File API

2.2.1 File header API

```
struct File_head *file_head_new();

Status file_head_ctor(struct File_head *const, FILE *const);
Status file_head_dtor(struct File_head *);

Status file_head_read(struct File_head *const header, const Fileoff foff,
const size_t sz, void *const o_data);
Status file_head_write(struct File_head *const header, const Fileoff foff,
const size_t sz, const void *const data);

Status file_head_sync(struct File_head *const);

Status file_head_shift_lsp(struct File_head *const, const size_t); // Last section ptr
```

2.2.2 File API

```
struct File *file_new();

Status file_ctor(struct File *const, FILE *const);
Status file_dtor(struct File *);

Status file_add_type(struct File *const file, const Type *const type, Fileoff *wrt_foff);
Status file_delete_type(struct File *const file, const String *const name);
Status file_find_type(struct File *const file, const String *const name, Type *const o_type,
Sectoff *const o_adr, struct Sect_types **o_sect_ptr);
Status file_read_type(struct File *const file, const Fileoff foff, Type *const o_type);

Status file_write(struct File *const file, const Json *const json,
Fileoff dad_fileoff, Fileoff type_foff, Fileoff *const wrt_foff);

Status file_read(struct File *const file, const Fileoff fileoff,
Json *const ret_json, Entity* const ret_entity);

Status file_delete(struct File *const file, const Fileoff fileoff,
const bool is_root);

Status file_update(struct File *const file, const Fileoff fileoff,
const Json *const new_json, const Fileoff dad_ptr,
const Fileoff type_ptr, bool is_root, Fileoff *cur_fileoff);

Status file_find(struct File *const file, struct Sect_ext *section,
const Query *const query, List_Pair_Json_Entity *const o_obj_col);
```

2.3 User API

```
struct File *user_open_file(const char *const name);
Status user_close_file(struct File *const file);

Status user_read_type(struct File *const file, const String *const name, Type *const o_type);
Status user_add_type(struct File *const file, const Type *const type);
Status user_delete_type(struct File *const file, const String *const name);

Status user_write(struct File *const file, const Json *const json, const String *const type_name);
struct Iter *user_read(struct File *const file, Query *const query);
Status user_update(struct File *const file, Query *const query, const Json *const new_json);
Status user_delete(struct File *const file, Query *const query);
```

3 Аспекты реализации

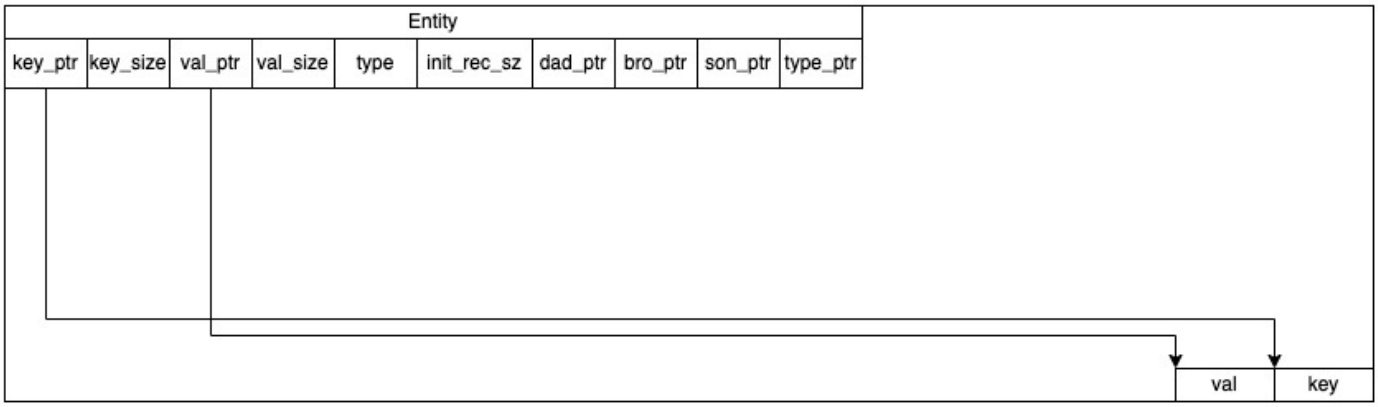
3.1 Физическая реализация базы данных

Файл для размещения данных использует секции, секции представляют из себя куски файла размером 8192 байта.

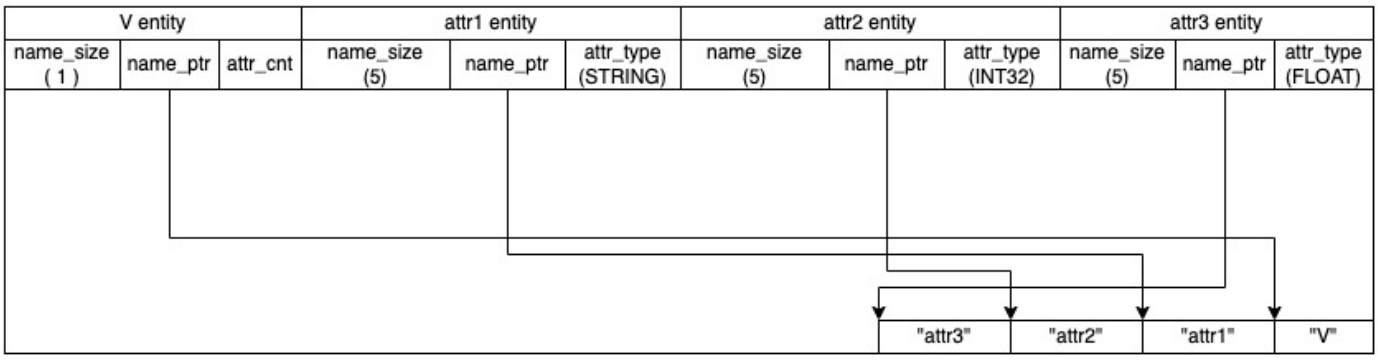
У каждой секции есть header, который хранит метаинформацию для обслуживания.

```
typedef struct Sect_head
{
    size_t free_space;
    Fileoff next_ptr;    // Nullable
    Sectoff lst_itm_ptr; // Pointer to first free cell(after items)
    Sectoff fst_rec_ptr; // Pointer to last free cell(before records)
    Fileoff file_offset; // Section offset from file start
    FILE *filp;
} Sect_head;
```

Размещение нод происходит по следующей логике:



Размещение типов происходит по следующей логике:



Далее мы работаем с этими секциями через file.

У него тоже есть свой header. Который хранит метаинформацию.

```
typedef struct File_head
{
    Fileoff lst_sect_ptr; // Points first free bit after sects
    FILE *filp;
} File_head;
```

3.2 Section API

3.2.1 Реализация Write

- Сначала проверим влезет ли запись в секцию
- Двигаем first record ptr на sizeof(ключ) + sizeof(значение)
- Записываем ключ
- Записываем значение
- Записываем Entity
- Двигаем last item ptr на sizeof(Entity)

3.2.2 Реализация Read

- Читаем Entity
- Читаем ключ по key_ptr из Entity
- Читаем значение по val_ptr из Entity

3.2.3 Реализация Delete

- Читаем удаляемую сущность
- Если сущность на границе то двигаем указатели
- Возвращаем удаляемую сущность при предоставлении буфера
- Очищаем байты записей
- Очищаем байты сущности
- Сжимаем указатели на lip и frp
- Синхронизируем header

3.2.4 Реализация Update

1. Считываем обновляемую сущность
2. Дотягиваем размер новой сущности до размера старой при необходимости
3. Если record находиться на границе
    - Если можем вставить запись в свободное пространство, делаем
4. Если record находится внутри
    - Если не можем вставить запись на место предыдущей записи вставляем в свободное место внутри секции, если хватает
    - Если можем, то вставляем на место предыдущей записи, прижимая запись к правой границе

3.3 File API

На уровне **File API** мы просто перебираем по очереди cur -> bro -> son и вызываем интересующий метод из **Section API**

3.4 User API

На уровне **User API** к выполнению добавляются типы и использование итератора для выполнения операций над группой элементов

К примеру код Delete:

```
iter_ctor(iter, file, query);

    int cnt = 0;

    while (iter_is_avail(iter))
    {
        DO_OR_FAIL(file_delete(file, iter_get_json(iter)->foff, true));
        cnt++;

        iter_next(iter);
    }
    iter_dtor(iter);
```

4 Результаты

Для создания программы было принято решение использовать TDD. Был создан каркас из тестов на секции и файловые операции с типами. Все исходники тестов лежат по ссылке

`https://github.com/dmittrey/low-level-programming/tree/main/solution/src/tests`

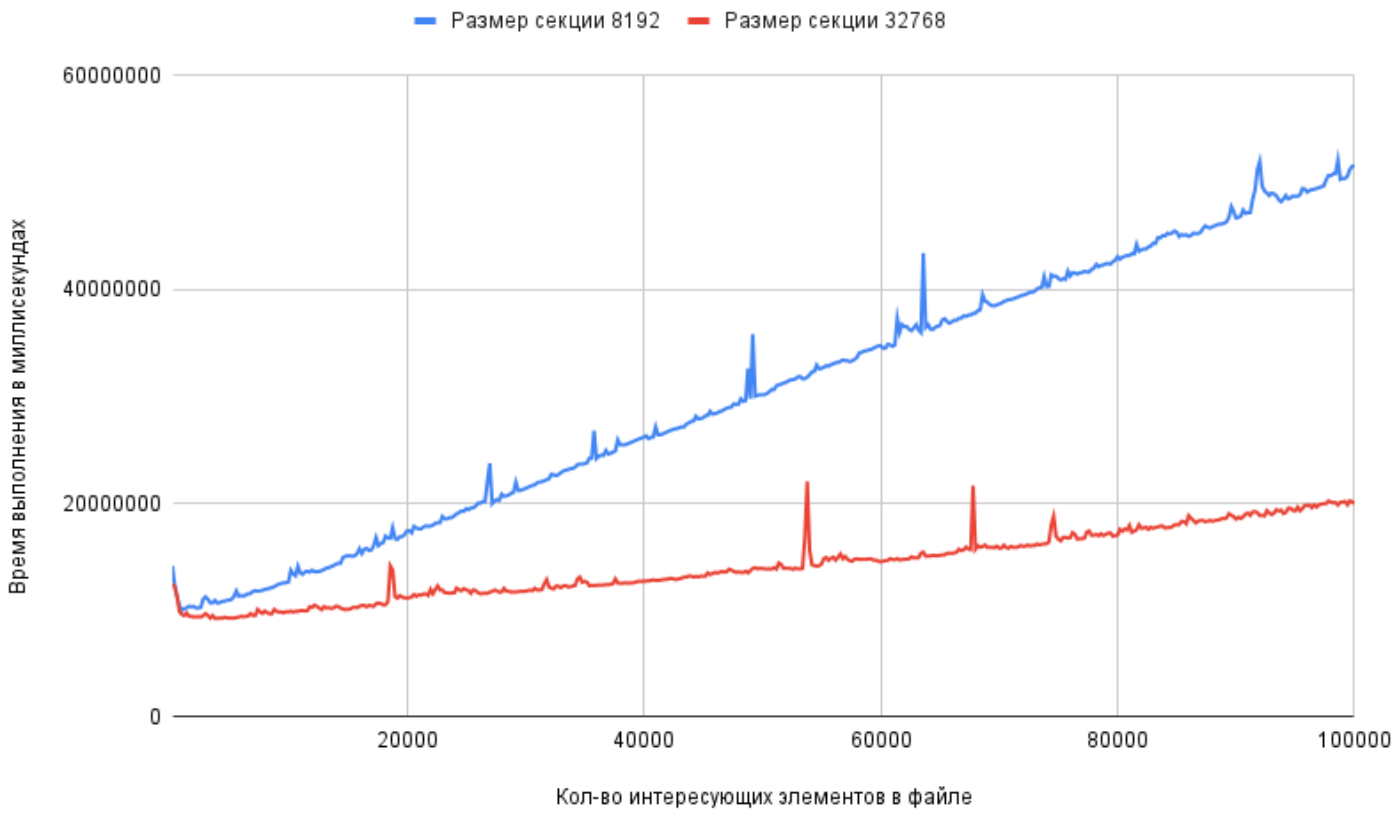
Использавалась практика AAA.

5 Выводы

В ходе долгой работы я написал и протестировал субд которая может работать с документным деревом.

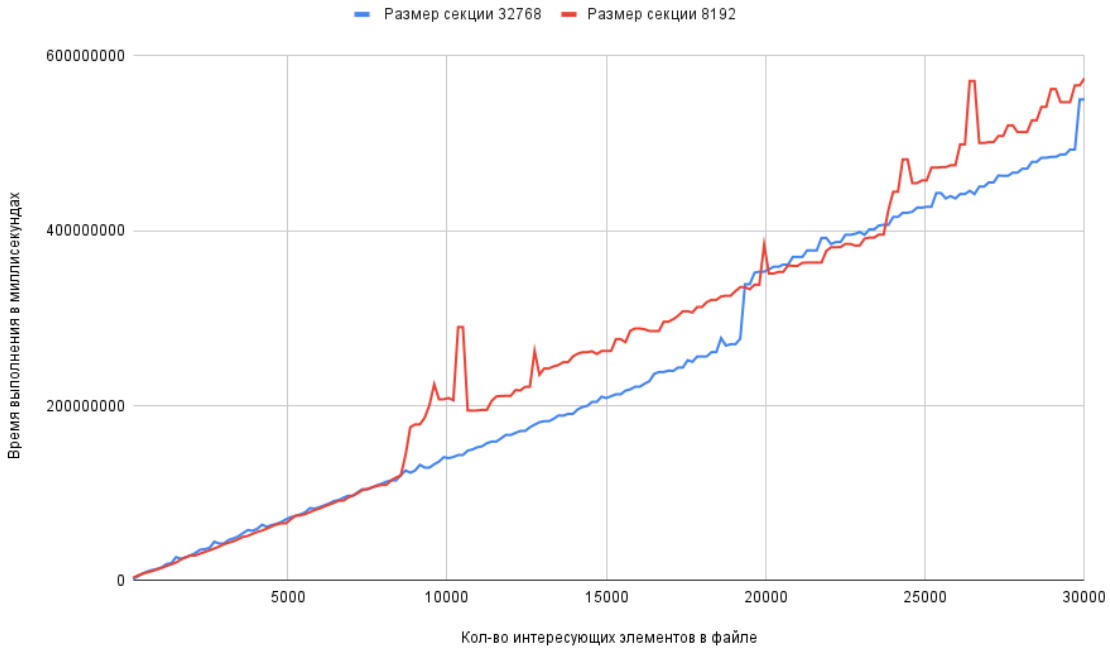
В ходе тестов я рассмотрел ассимптотики 4 операций:

- **Write**

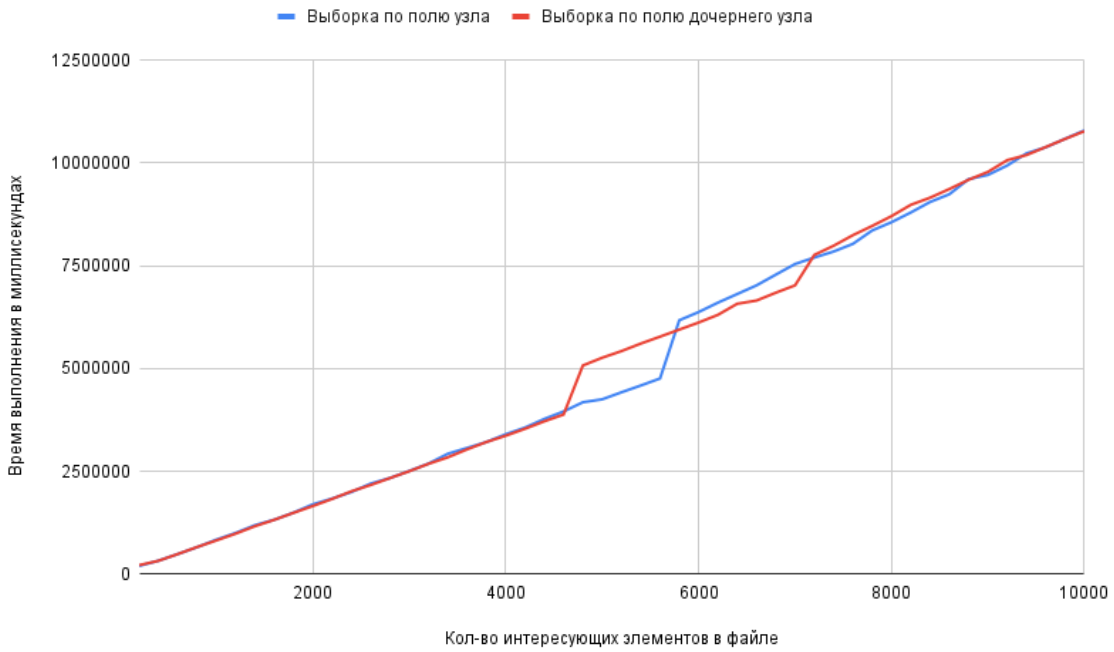


При вставке элемента в базу при размере секции в 8кб и 32кб видно, что время выполнения запроса растет линейно. При этом градус наклона линии зависит от кол-ва секций(Чем больше секция, тем меньше секций). То есть приходим к выводу что время выполнения растет линейно и заивисит от кол-ва секций.

- **Read**

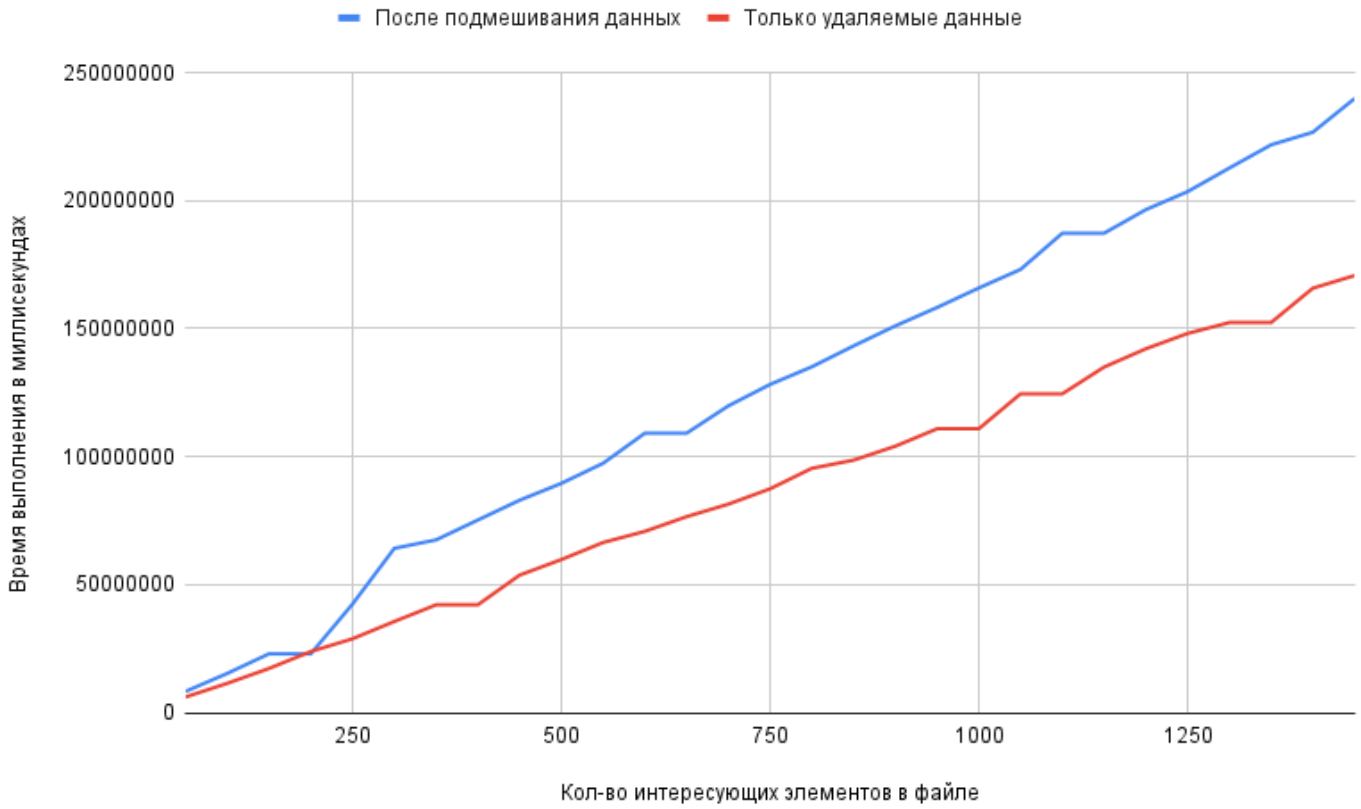


При чтении элемента из базы при размере секции в 8кб и 32кб видно, что время выполнения запроса растет линейно. При этом градус наклона линии никак не зависит от кол-ва секций(Чем больше секция, тем меньше секций). То есть приходим к выводу, что время выполнения растет линейно и заивисит от кол-ва интересующих элементов в базе.



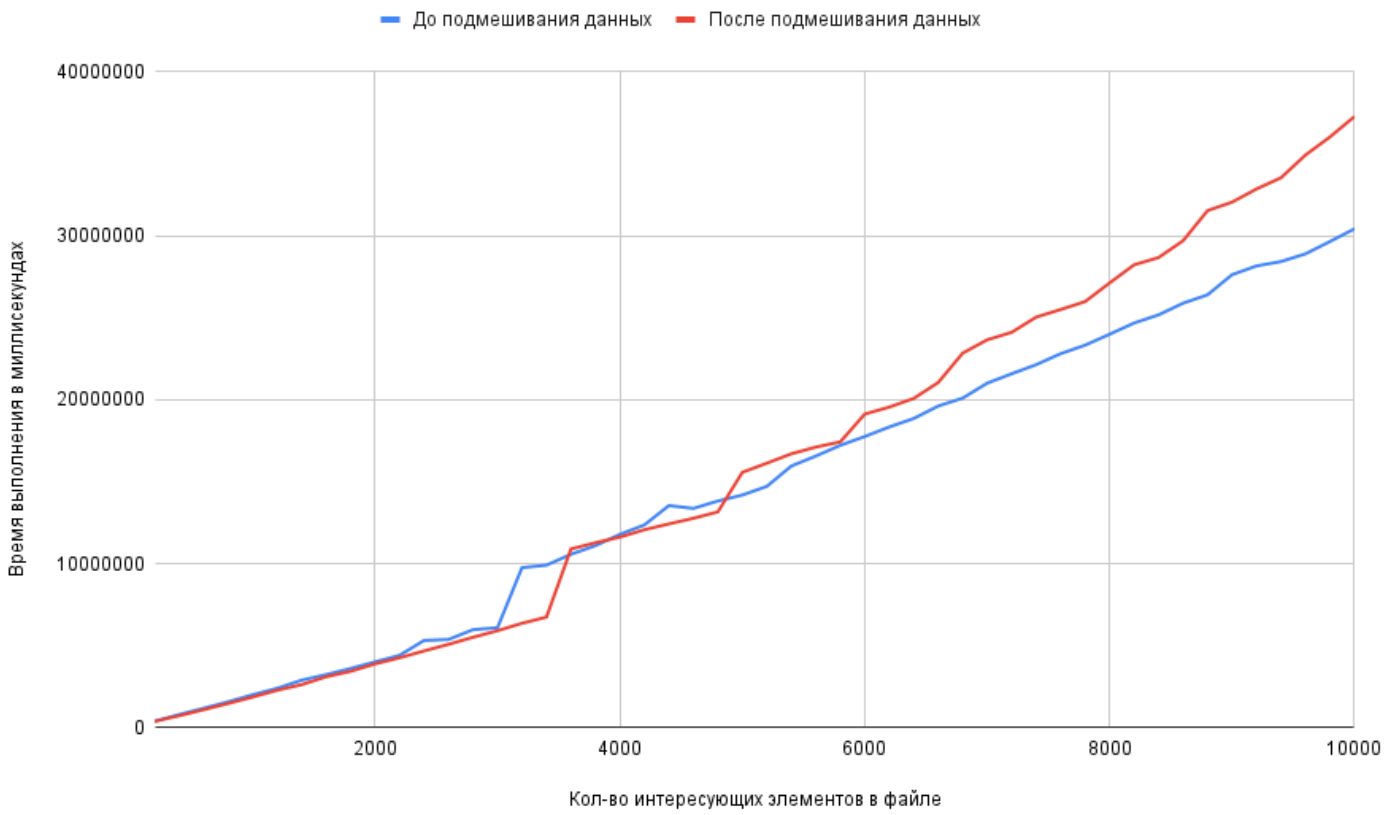
В данном случае мы ищем элемент по атрибуту и по атрибуту дочернего узла. Существенного изменения времени выполнения не наблюдается. Видно только изменение времени в середине. Смею предположить что на границе секций был разрыв. Некоторые дочерние ноды попали в первую, некоторые во вторую и интересующий дочерний узел попал во вторую, а атрибут был записан в первую секцию и мы прочитали из нее(синяя под красной). То есть чтение зависит от кол-ва целевых элементов.

- **Delete**



Удаление также зависит от кол-ва целевых элементов. При подмешивании элементов в базу, мы видим существенное увеличение времени выполнения, но тоже за линию. То есть я доказываю, что время удаления не зависит от кол-ва всех элементов.

- **Update**



При обновлении графики несущественно отличаются. Дело в том, что я обновляю в зависимости от кол-ва затрагиваемых элементов и поэтому заметного снижения скорости обновления не наблюдается.