Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Низкоуровневому программированию»

Выполнил:

Студент группы P33302
Верзаков А.Ю.

Преподаватель:
Кореньков Ю.Д.

Санкт-Петербург

2023

Задание:

Создать модуль, реализующий хранение в одном файле данных (выборку, размещение и гранулярное обновление) информации общим объёмом от 10GB соответствующего варианту вида.

Описание:

Программа состоит из нескольких модулей:

Основные:

- OSFile – базовые операции над файлом (запись, чтение, открытие, закрытие)
- DataFile – низкоуровневые операции для работы с файлом (запись вершин, строк, и тд.)
- StorageFileInnerAPI – операции добавления, выборки, удаления и обновления элементов дерева
- StorageFilePublicAPI – публичный интерфейс для работы с файлом в интерактивном режиме

Вспомогательные:

- UserAPI – интерфейс для работы с программой в интерактивном режиме
- Utils – вспомогательные методы для работы со строками, инициализации файла и т. п. внутри программы.

Реализованные структуры:

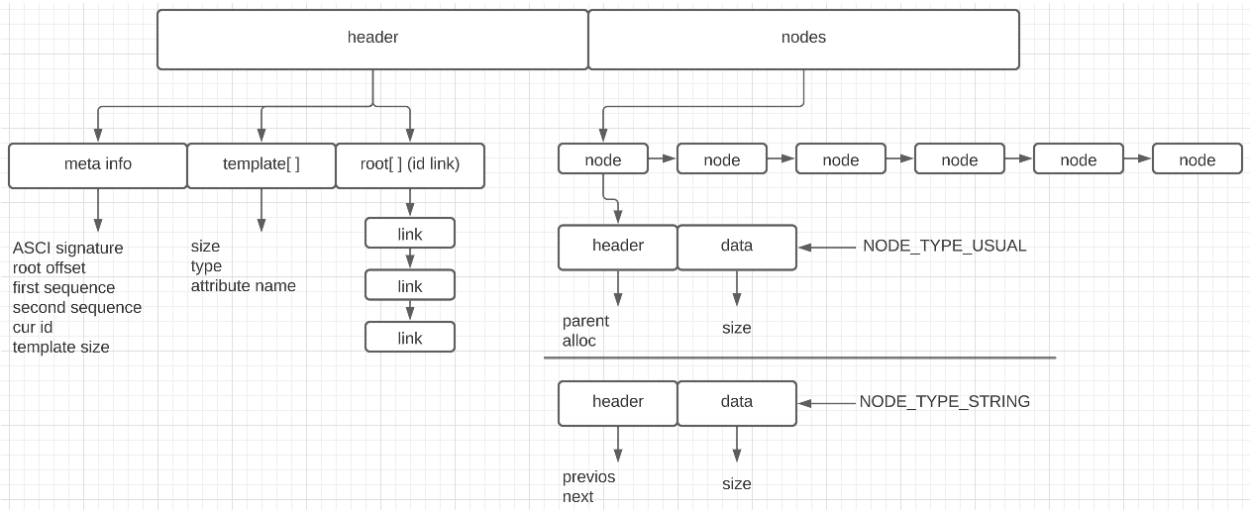
```
#pragma once
#include <inttypes.h>

#define NODE_HEAD_SIZE sizeof(union nodeHeader)
#define NODE_SIZE sizeof(struct keyNode)

union nodeHeader {
    struct {
        uint64_t parentNode;
        uint64_t allocNode;
    };
    struct {
        uint64_t prevNode;
        uint64_t nextNode;
    };
};

struct keyNode {

    union nodeHeader header;
    uint64_t* data;
};
```

```c
#pragma once
#include <inttypes.h>

#define META_SIZE sizeof(struct treeMeta)
#define ATTR_HEAD_SIZE sizeof(struct attributeHeader)
#define ATTR_INFO_SIZE sizeof(struct nodeAttributeInfo)
#define SCHEMA_SIZE sizeof(struct treeSchema)

#define INT 0
#define FLOAT 1
#define STRING 2
#define BOOL 3

struct treeMeta {
    uint64_t ASCIISign;
    uint64_t rootOffset;
    uint64_t firstSeq;
    uint64_t secondSeq;
    uint64_t curId;
    uint64_t templateSize;
};

#pragma pack(push, 4)
struct attributeHeader {
    uint32_t size;
    uint32_t type;
};
struct nodeAttributeInfo {
    struct attributeHeader* header;
    char* attributeName;
};
#pragma pack(pop)

struct treeSchema
{
    struct nodeAttributeInfo** nodesTemplate;
    struct treeMeta* meta;
    uint64_t* root;
};
```

Модель данных:

Файл состоит из заголовка и множества вершин. Заголовок содержит шаблон полей и типов, который применяется ко всем вершинам, содержащимся в файле. Вершины делятся на 2 типа – обычные и строковые. Обычные вершины хранят в себе данные непосредственно, INT, FLOAT и BOOL – в явном виде, на тип STRING указывает ссылка – ссылка на строковый кортеж, который представляет из себя двусвязный список.

Схема файла:



Для работы с элементами данных были реализованы следующие операции:

- storageInsertNode – вставка элемента
- storageRetriveNode – выборка элемента (ов)
- storageUpdateNode – обновление элемента
- storageDeleteNode – удаление элемента

Также реализованы (публично) операции закрытия/открытия файла и вывод справки по всем доступным командам:

- storageCloseFile
- storageOpenOrCreateFile
- storageGetHelp

соответственно.

Примеры использования программы:

```
Initializing template.
Input the number of attributes in nodes: 2
<---- Attribute 0   ---->
Attribute name: phone
0. Integer type
1. Float type
2. String type
3. Boolean type
Choose field type: 2
<---- Attribute 1   ---->
Attribute name: model
0. Integer type
1. Float type
2. String type
3. Boolean type
Choose field type: 0
File opened successfully!
Type 'help' for available commands info.
add 0 phone=iphone model=14
Node added
print nodes
<----- Node   0 ----->
Attribute: phone                ; Value: iphone;
Attribute: model                ; Value: 14;
```

Печать элементов:

```
<----- Node 6659 ----->
Attribute: phone                ; Value: iirbgncikg;
Attribute: model                ; Value: 63038;
<----- Node 6660 ----->
Attribute: phone                ; Value: flyfheojig;
Attribute: model                ; Value: 66250;
<----- Node 6661 ----->
Attribute: phone                ; Value: hbjtz;
Attribute: model                ; Value: 67396;
<----- Node 6662 ----->
Attribute: phone                ; Value: churcgrls;
Attribute: model                ; Value: 20106;
<----- Node 6663 ----->
Attribute: phone                ; Value: cwfjq;
Attribute: model                ; Value: 92338;
<----- Node 6664 ----->
Attribute: phone                ; Value: yfemk;
Attribute: model                ; Value: 49383;
<----- Node 6665 ----->
Attribute: phone                ; Value: dkcbfpizamsjg;
Attribute: model                ; Value: 25900;
```

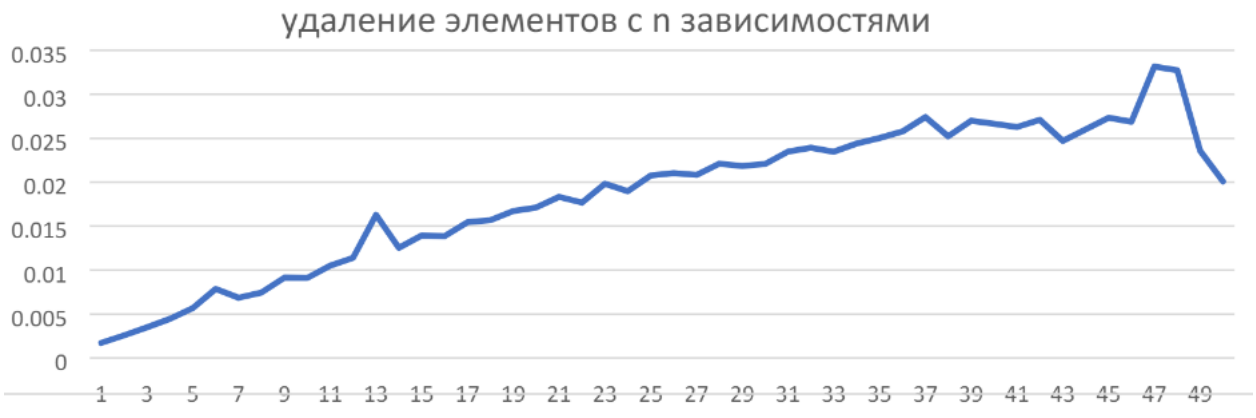
```
print schema
<----- Tree schema ----->
Current ID:          6666
Template size:       2
<----- Template ----->
Key    8 [Type    2]: phone
Key    8 [Type    0]: model
find_by id 1234
phone                : ereevpxnvopotpyy
model                : 90696
find by field phone ereevpxnvopotpyy
Unknown command, try using 'help'
find_by field phone ereevpxnvopotpyy
<----- Result set ----->
id: 1234
```

Графики производительности программы:

По времени:


Поиск элемента по полю
Время выполнения
0.005
0.004
0.003
0.002
0.001
0
1 133 265 397 529 661 793 925 1057 1189 1321 1453 1585 1717 1849 1981 2113 2245 2377 2509 2641 2773 2905 3037 3169 3301 3433 3565 3697 3829 3961 4093 4225 4357 4489 4621 4753 4885
Номер элемента


Обновление элемента по id
Время выполнения
0.0003
0.00025
0.0002
0.00015
0.0001
0.00005
0
1 137 273 409 545 681 817 953 1089 1225 1361 1497 1633 1769 1905 2041 2177 2313 2449 2585 2721 2857 2993 3129 3265 3401 3537 3673 3809 3945 4081 4217 4353 4489 4625 4761 4897
Номер элемента


удаление элементов с n зависимостями
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49


Поиск элемента по родителю (джоин)
0.012
0.01
0.008
0.006
0.004
0.002
0
1
27
53
79
105
131
157
183
209
235
261
287
313
339
365
391
417
443
469
495
521
547
573
599
625
651
677
703
729
755
781
807
833
859
885
911
937
963
989
1015

По памяти:


зависимость размера файла от количества элементов
6000000
5000000
4000000
3000000
2000000
1000000
0
размер
1
279
557
835
1113
1391
1669
1947
2225
2503
2781
3059
3337
3615
3893
4171
4449
4727
5005
5283
5561
5839
6117
6395
6673
6951
7229
7507
7785
8063
8341
8619
8897
9175
9453
9731
количество


Использование оперативной памяти при добавлении элементов
200
150
100
50
0
занятая память
1
60
119
178
237
296
355
414
473
532
591
650
709
768
827
886
945
1004
1063
1122
1181
1240
1299
1358
1417
1476
1535
1594
1653
1712
1771
1830
1889
1948
2007
2066
2125
2184
количество элементов

Вывод:

В ходе работы я разработал программу, которая поддерживает хранение и обработку информации в файле большого объема. Также был написан консольный интерфейс для взаимодействия с файлом. Были проведены тесты, которые удовлетворяют требованиям и показывают, что программа потребляет оптимальное количество памяти и времени.