Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Лабораторная работа №1

По дисциплине

«Низкоуровневое программирование»

Вариант №3

Группа: P33302

Выполнил: Бахтеев Богдан

Преподаватель: Кореньков Юрий Дмитриевич

Санкт-Петербург

декабрь 2023

Цель

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

Порядок выполнения

1 Спроектировать структуры данных для представления информации в оперативной памяти

a. Для порции данных, состоящий из элементов определённого рода (см форму данных), поддержать тривиальные значения по меньшей мере следующих типов: четырёхбайтовые целые числа и числа с плавающей точкой, текстовые строки произвольной длины, булевские значения

b. Для информации о запросе

2 Спроектировать представление данных с учетом схемы для файла данных и реализовать базовые операции для работы с ним:

a. Операции над схемой данных (создание и удаление элементов схемы)

b. Базовые операции над элементами данных в соответствии с текущим состоянием схемы (над узлами или записями заданного вида)

i. Вставка элемента данных

ii. Перечисление элементов данных

iii. Обновление элемента данных

iv. Удаление элемента данных

3 Используя в сигнатурах только структуры данных из п.1, реализовать публичный интерфейс со следующими операциями над файлом данных:

a. Добавление, удаление и получение информации о элементах схемы данных, размещаемых в файле данных, на уровне, соответствующем виду узлов или записей

b. Добавление нового элемента данных определённого вида

c. Выборка набора элементов данных с учётом заданных условий и отношений со смежными элементами данных (по свойствам/полями/атрибутам и логическим связям соответственно)

d. Обновление элементов данных, соответствующих заданным условиям

e. Удаление элементов данных, соответствующих заданным условиям

4 Реализовать тестовую программу для демонстрации работоспособности решения

a. Параметры для всех операций задаются посредством формирования соответствующих структур данных

b. Показать, что при выполнении операций, результат выполнения которых не отражает отношения между элементами данных, потребление оперативной памяти стремится к O(1) независимо от общего объёма фактического затрагиваемых данных

c. Показать, что операция вставки выполняется за O(1) независимо от размера данных, представленных в файле

d. Показать, что операция выборки без учёта отношений (но с опциональными условиями) выполняется за O(n), где n – количество представленных элементов данных выбираемого вида

e. Показать, что операции обновления и удаления элемента данных выполняются не более чем за $O(n*m) > t \rightarrow O(n+m)$, где n – количество представленных элементов данных обрабатываемого вида, m – количество фактически затронутых элементов данных

f. Показать, что размер файла данных всегда пропорционален размещённых элементов данных

g. Показать работоспособность решения под управлением ОС семейств Windows и *NIX

5 Результаты тестирования по п.4 представить в составе отчёта, при этом:

a. В части 3 привести описание структур данных, разработанных в соответствии с п.1

b. В части 4 описать решение, реализованное в соответствии с пп.2-3

c. В часть 5 включить графики на основе тестов, демонстрирующие амортизированные показатели ресурсоёмкости по п. 4

Примеры работы программы

```
FIND ALL ELEMENTS
id - 1
connected nodes:
integer - 22
float - 22.20
bool - 1
string - Hello world

id - 2
connected nodes:
integer - 22
float - 22.20
bool - 1
string - Hello world

ADD ELEMENT
FIND ALL ELEMENTS
id - 1
connected nodes:
integer - 22
float - 22.20
bool - 1
string - Hello world
```

```
id - 2
connected nodes:
integer - 22
float - 22.20
bool - 1
string - Hello world

id - 3
connected nodes:
integer - 33
float - 22.20
bool - 1
string - Hello world

FIND BY ID 2
id - 2
connected nodes:
integer - 22
float - 22.20
bool - 1
string - Hello world
```

Аспекты реализации

В файле хранится структура с мета информацией

```
struct file_info {
    uint64_t id_seq;
    uint64_t link_to_first_element;
    uint64_t link_to_last_element;
    uint64_t link_to_next_free_memory;
};
```

1) Последовательность id
2) Отступ в файле до 1 элемента
3) Отступ до последнего
4) Следующее место для записи

```
struct node {
    uint64_t id;
    uint64_t nodes[MAX_NEIGHBOURS];
    uint64_t int_number;
    uint64_t float_number;
    uint64_t bool_number;
    uint64_t string_size;
    uint64_t link_to_string;
    uint64_t link_to_prev_node;
};

struct node_list {
    struct node *node;
    struct node_list *prev;
};
```

Структура, хранящая сам элемент. Из особенного в ней строка. А именно, то, что из-за разно размерности строк мы не можем хранить её напрямую в node. Вместо этого мы хранит отступ до неё с начала файла. И при необходимости будем считывать.

Так же некоторые crud функции могут возвращать не один элемент, поэтому реализована структура node_list – односвязный список.

Сама же node хранит ссылку на предыдущую, для удобной итерации.

Так же для борьбы с фрагментацией в файле при удалении мы переставляем последний элемент на место удаленного.

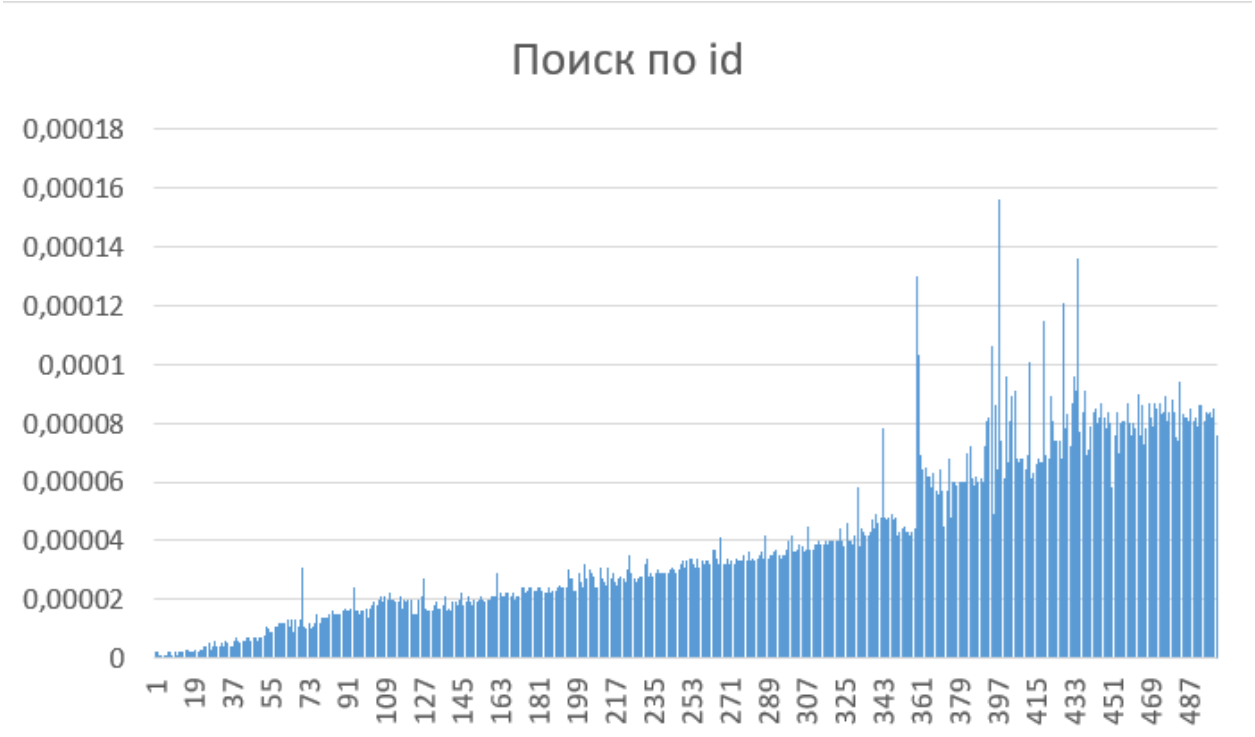
Операции:

- Добавление – мы записываем новую node в конец файла. И кладем в неё ссылку на предыдущую.
- Поиск по id – в мета информации хранится ссылка на последнюю node, мы берем её и с помощью ссылок в самих nodes итерируемая по всем элементам, и находим нужный.
- Удаление по id – ищем нужную node в файле, если есть, то берем последнюю node и вставляем её вместо удаляемой, предварительно поменяв ссылки в последней на те, что были в удаленной.
- Соединение двух node’s по id – ищем эти две node’s в файле, читаем их. Связываем и записываем обратно.
- Обновление поля по id – ищем node по id, и заменяем значение нужного нам поля на новое, записываем node обратно.
- Поиск по значению поля – бежим по всему файлу и ищем node’s с нужным нам значением поля, кладем их в node_list, ибо их может быть и не одна штука.

Результаты

Семантика операций:

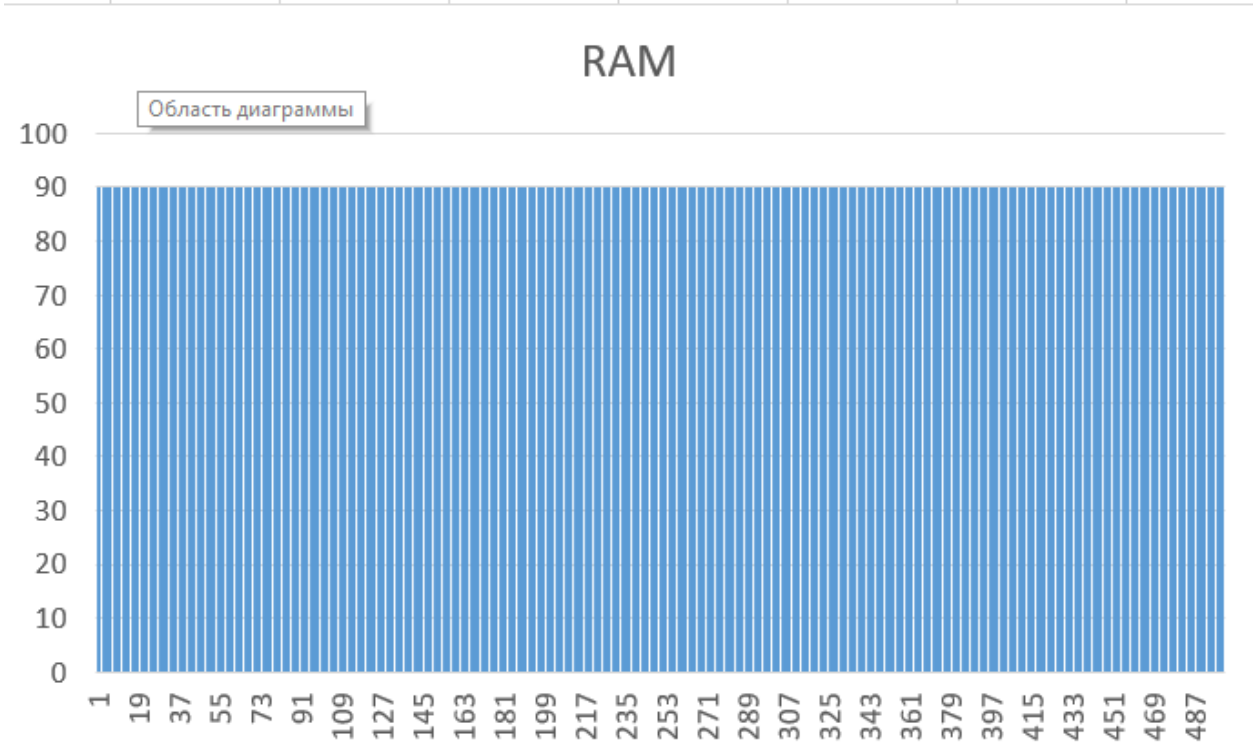
- Добавление O (1)
- Поиск по id O (n)
- Удаление по id O (n * m), где n весь файл, а m – фактически затронутые
- Обновление поля по id O (n)
- Поиск по значению поля O (n)


Поиск по id
0,00018
0,00016
0,00014
0,00012
0,0001
0,00008
0,00006
0,00004
0,00002
0
1
19
37
55
73
91
109
127
145
163
181
199
217
235
253
271
289
307
325
343
361
379
397
415
433
451
469
487


Обновление по id
0,00014
0,00012
0,0001
0,00008
0,00006
0,00004
0,00002
0
1
20
39
58
77
96
115
134
153
172
191
210
229
248
267
286
305
324
343
362
381
400
419
438
457
476
495

Без учета зависимых

Замера размера файла происходила с помощью утилиты gnome-system-monitor



Вывод:

В результате работы был создан модуль, реализующий хранения доступ и модификацию данных внутри файла.