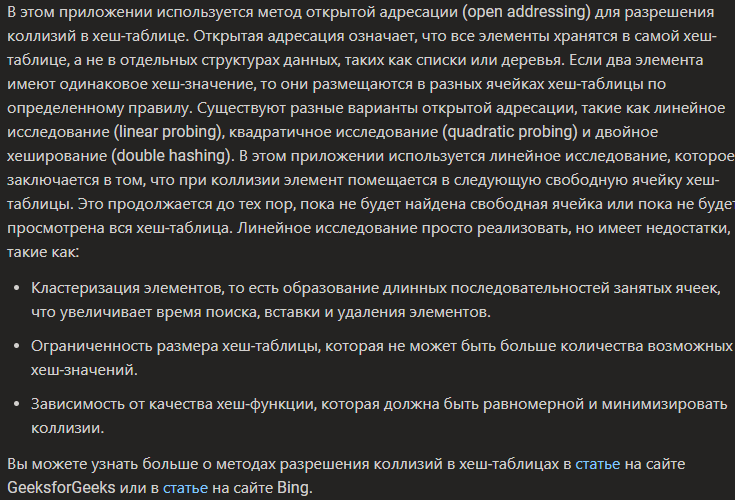
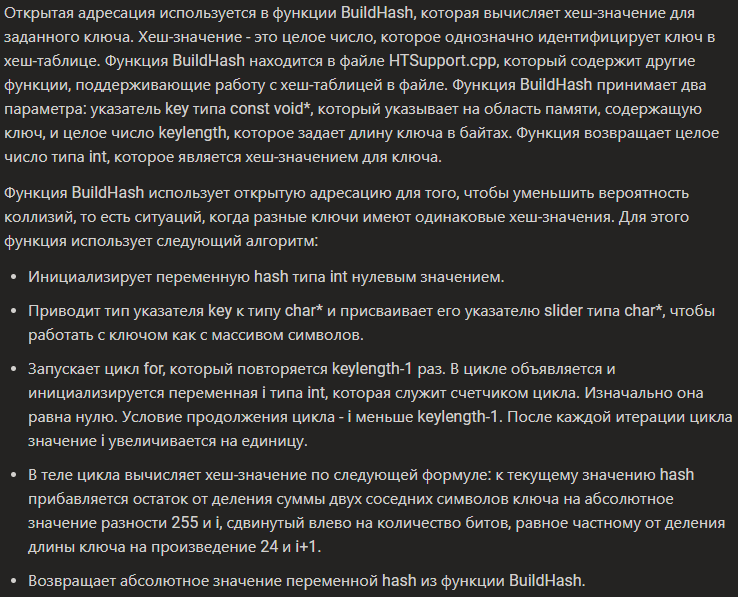
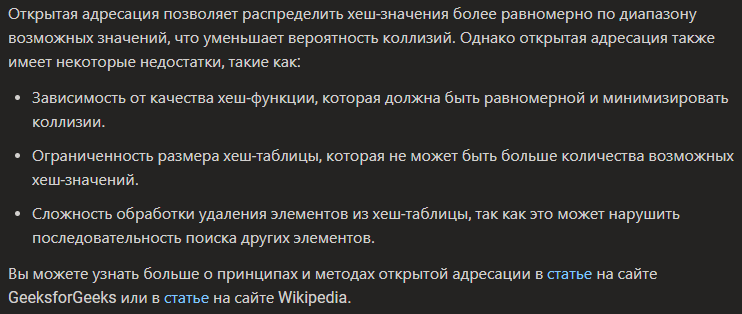
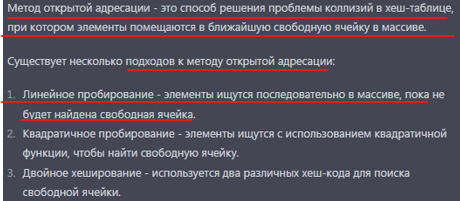
**Лабораторная №10. File mapping – Файловая проекция**









***Файловые проекции: что это и для чего***

В windows OC поддерживают механизм работы с файлами через файловые проекции, механиз подразумевает что файл проецируется в виртуальное адресное пространство процесса.

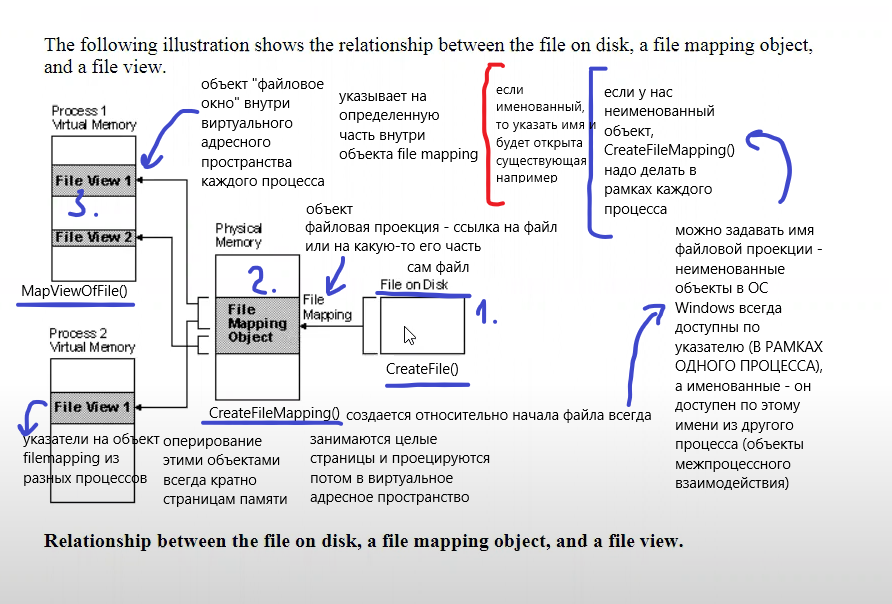
То есть работа с файлом в конечном счете будет осуществляться через указатель на память(lpvoid,int\* и тд).

Механизм изначально был предназначен для того чтобы открывать файл большого размера и случайным образом переходить к его определенным местам (не перемещая setpointer и тд)

Механизм позволяет проецировать файлы как на реальные файлы так и swap file (несуществующий).

Это может использоваться как ipc процессов (процесс создавший проекцию может записывать, а другой читать). Может использоваться как канал взаимодействия между процессами (для обмена инфы).

Файловые проекции позволяют работать с памятью (адресным пространством), а не с файлом.



Есть файл на диске, файловая проекция это File mapping.

**Файловая проекция** – это ссылка на весь файл или на некоторую часть, которая проецируется в физическую память. То есть вот эта часть данных будет находится в физической памяти.

Далее для того чтобы получить доступ к File Mapping объекту (он может быть довольно таки большим 50-100мб), необходимо создавать «файловое окно» File View (файловое представление).

Это уже объект внутри каждого виртуального адресного пространства процесса, который является указателем на определенную часть File Mapping объекта

Оперирование этими объектами (file maping, file view) кратно страницам памяти. Потому что занимаются целые страницы и проецируются в виртуальное адресно пространство процесса.

Страницы памяти - это части физической памяти компьютера, которые используются

для хранения данных, которые используются процессами в операционной системе. Эти

страницы являются фиксированными блоками памяти фиксированного размера,

например, обычно размер страницы равен 4 КБ (4096 байт) в большинстве

современных компьютеров.

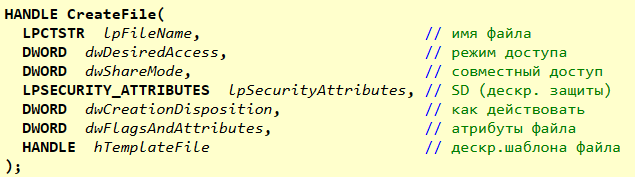
Когда процесс запрашивает выделение памяти, операционная система выделяет

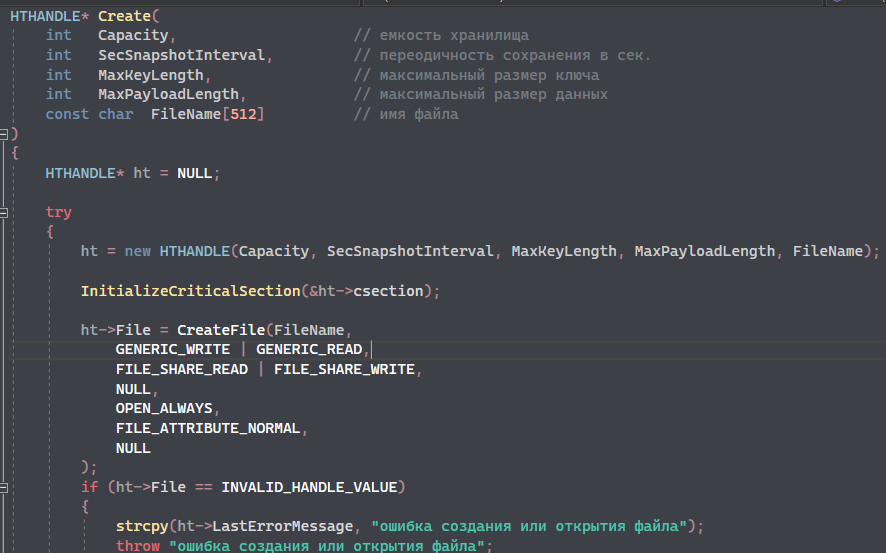
необходимое количество страниц памяти. Данные на этих страницах могут быть

прочитаны и записаны процессом, который владеет этой страницей памяти.

***OS10\_01***

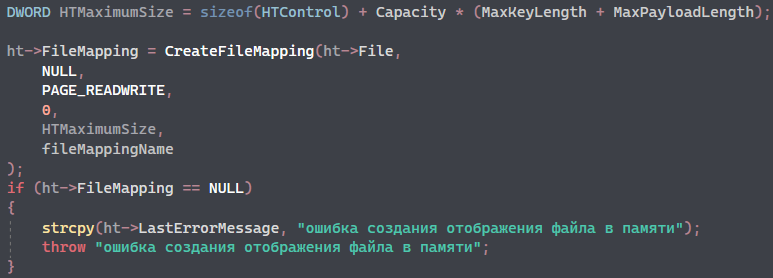
1. **Создать файл CreateFile**

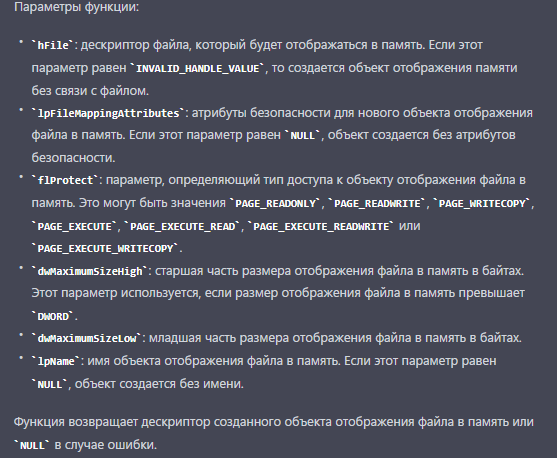
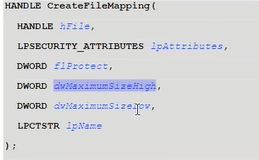


****

1. **Создать проекцию файла CreateFileMapping**

**CreteFileMapping подразумевает что файл должен быть открыт в настоящий момент и у нас должен быть Handle на этот файл.**

****

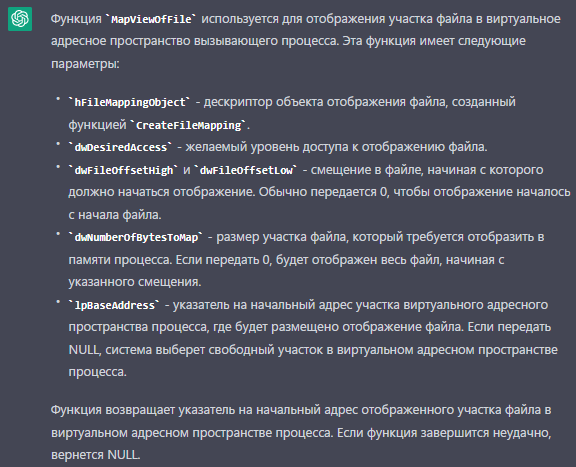


**Дескриптор, атрибуты, режим доступа к файловой проекции,**

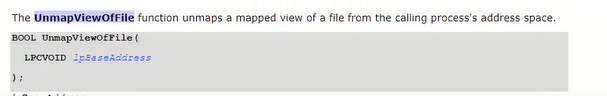
**размер файловой проекции (СОЗДАЕТСЯ ВСЕГДА ОТНОСИТЕЛЬНО НАЧАЛА ФАЙЛА):**

**максимальный размер файловой проекции (тк файлы могут быть большие, если мы создаем файл мапп то это не значит что она сразу будет загружена в память, только нужную часть файла) максимальный размер который будет доступен задается с помощью 2 значений.**

**lpName – имя файловой проекции**



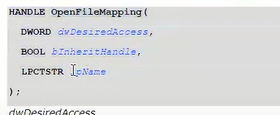
**Если FileView больше не нужна:**



**Указатель на начало памяти в адресном пространстве**

**Отменяет представление файла, очищает адресное пространство процесса.**

**Открыть существующую файловую проекцию:**

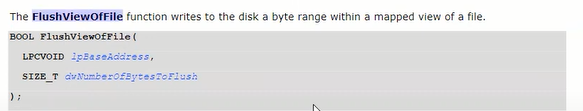


**Передаем имя существующего меппинга**

**Остается вопрос:**

**Мы получили указатель на память (MapViewOfFile), мы оттуда что-то пишем, они пишутся в память, как указать другому процессу что запись закончена можешь читать их и тд. Надо быть увереными что данные туда реально записаны:**

**на такую ситуацию есть функция FlashViewOfFile**





**Адрес смещения, размер байтов которые нужно записать.**

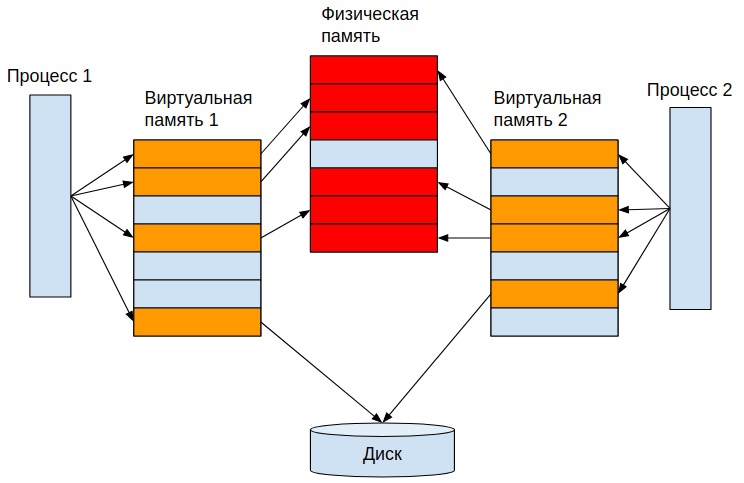
**Принудительно данные из виртуального адресного пространства записывает в файл и тем самым, если кто-то обратиться к памяти то он получит новые данные.**

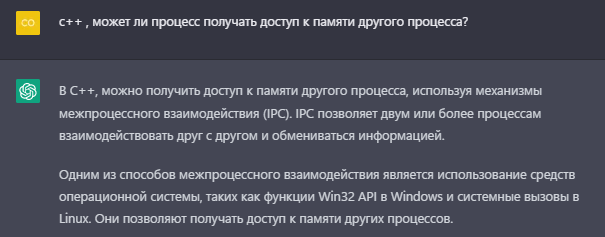
***Итог:***

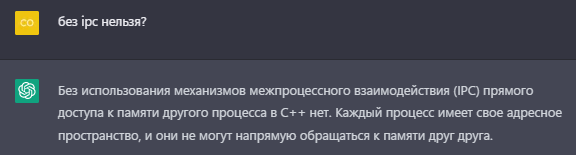
**Для того чтобы работать через файловую проекцию, нужно создать ее и споецировать в виртуальное адресное пространство процесса, где мы получим указатель на память, дальше мы работаем с данными как с памятью, *обращаемся как к массиву и тд*.**

**Где применяется: обмен данными между процессами, через обычные файлы не вариант тк мы будем реально обращаться к файлам на жестком диске, а через файловую проекцию мы обращаемся через память и обмен данными осуществляется В РАЗЫ БЫСТРЕЕ.**

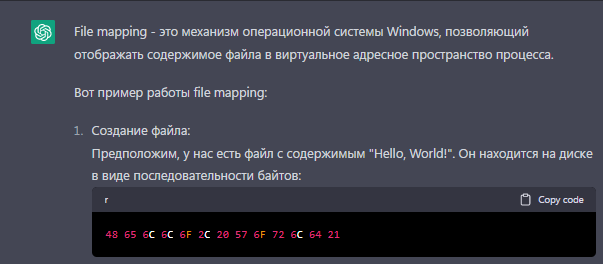
**Это очень помогает при работе с огромными файлами, такими как файлы базы данных, поскольку не весь файл должен находиться в физической памяти.**

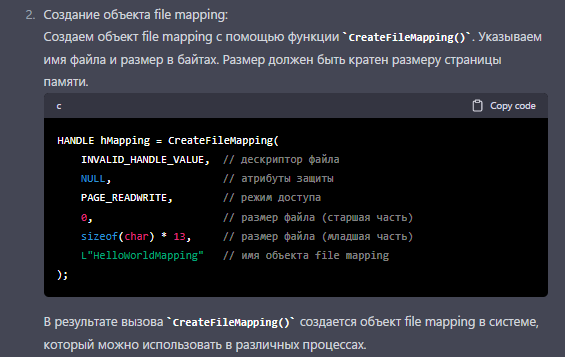
=====================================================================

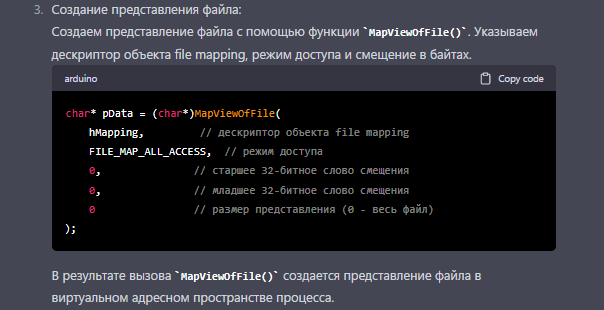


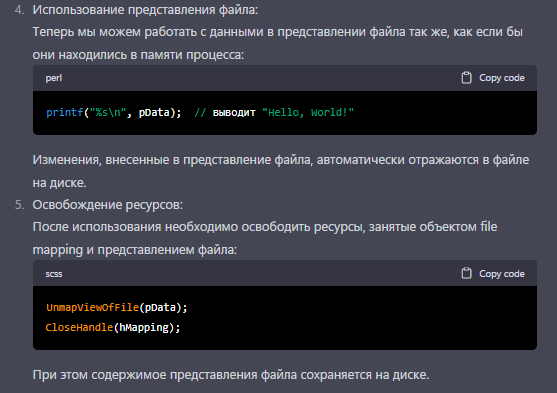


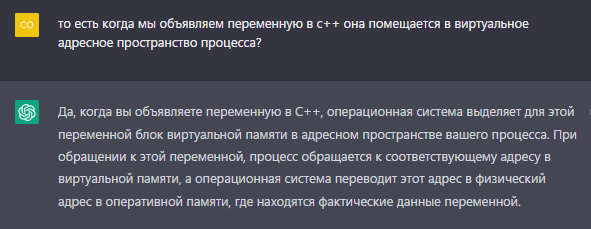
***Более просто про FileMapping***



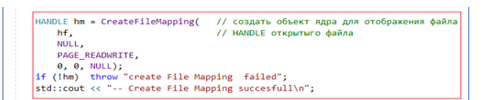


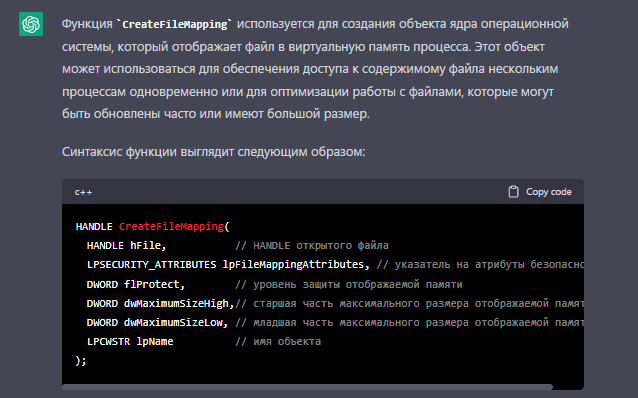


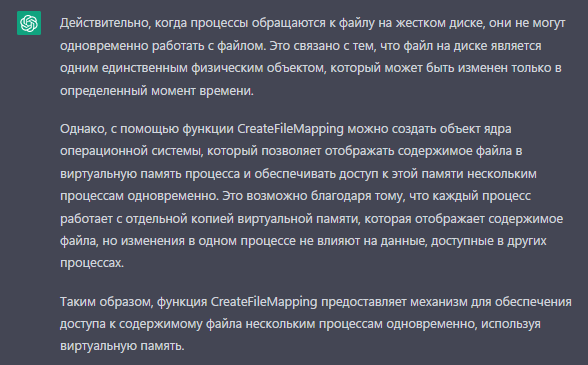


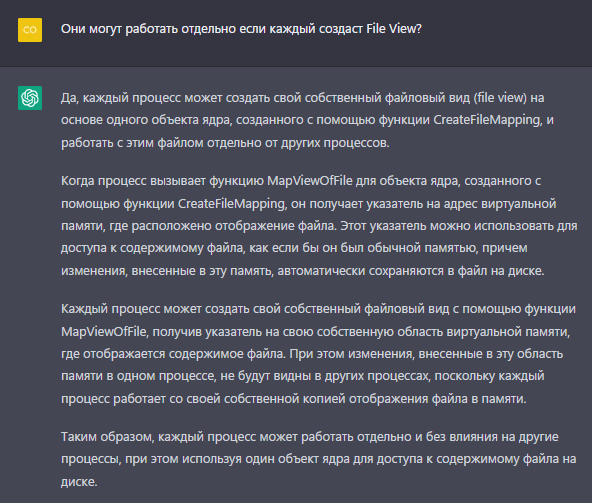


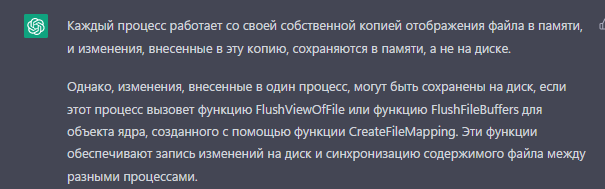
##### **CreateFileMapping**



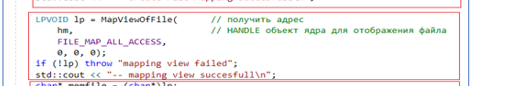


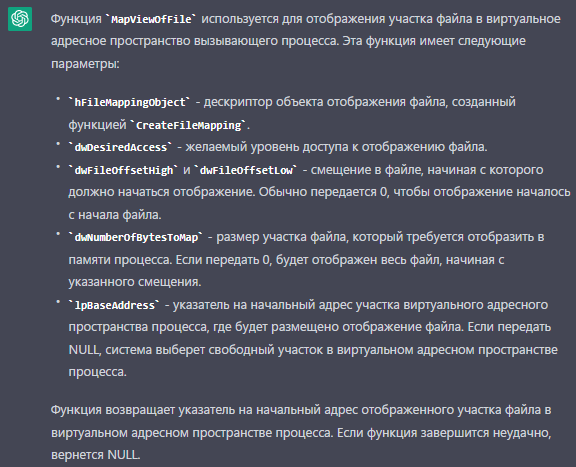


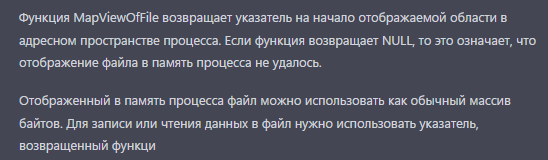




##### **MapViewOfFile**

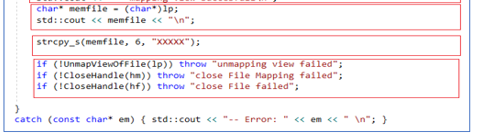


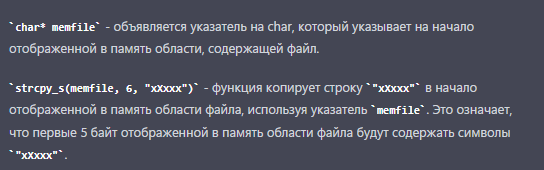


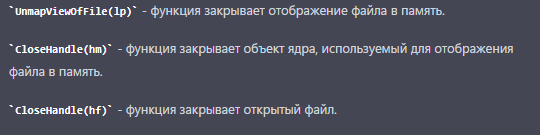


**Для записи и чтения используем указатель**

##### **Вывод файла и освобождение**





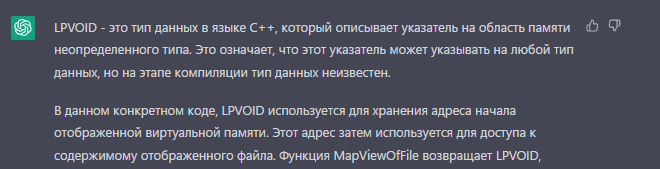


**Как я понял из-за unmapviewoffile происх запись в файл?**

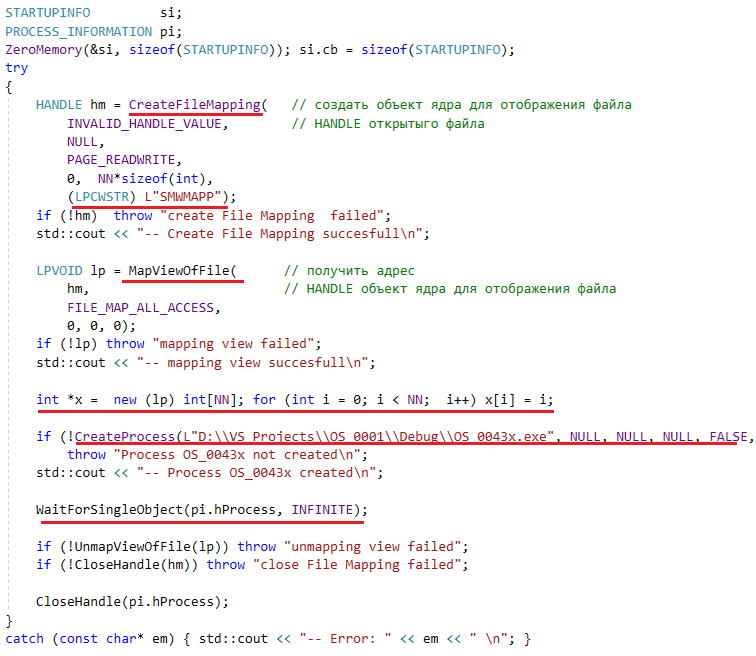
##### **===Что происходит с дескриптером после закрытия?**

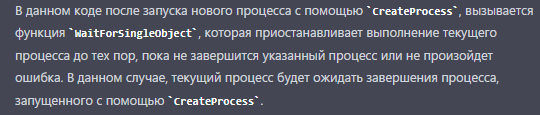


**LPVOID**



#### **OS: INVALID\_HANDLE\_VALUE , NAME FileMapping**

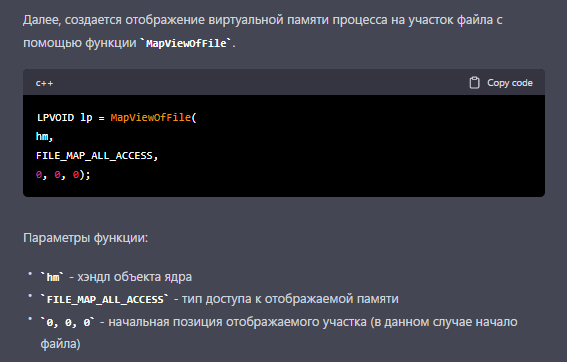
****

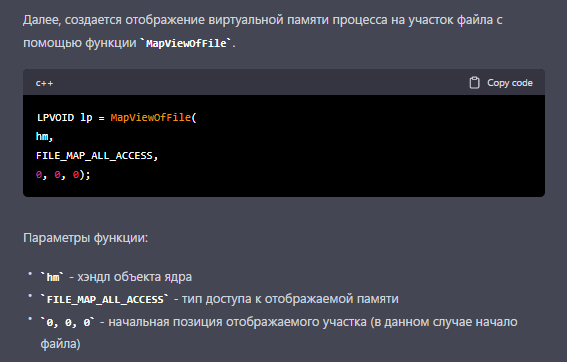


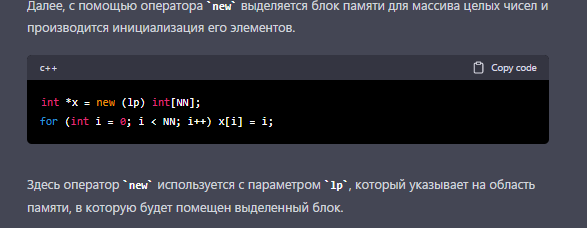
**После заверш второго идет овсвобождение viewoffile первого и closehandle**

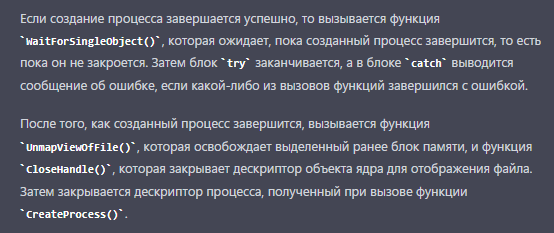




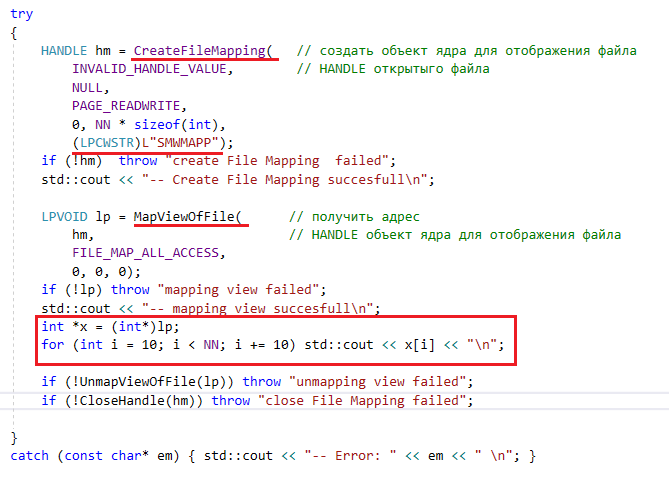


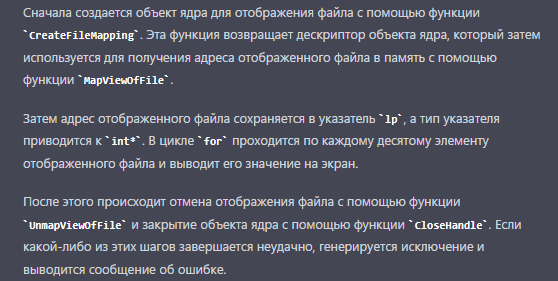




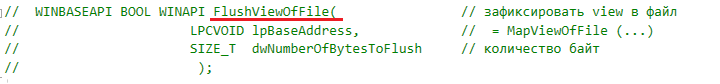


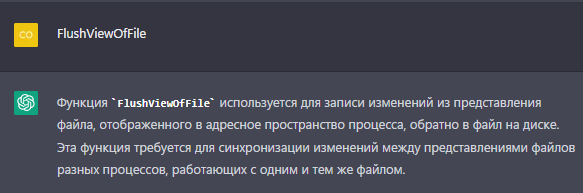
##### **Из другого процесса:**

****



#### **OS:FlushViewOfFile**

****



**Из адресного пространства -> файл**

### Метода + код лабы

#### **HT-хранилище (HT):**

**Программная система**, предназначенная для хранения данных в формате ключ/значение.

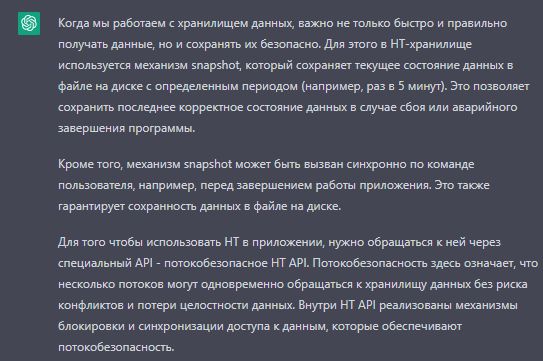
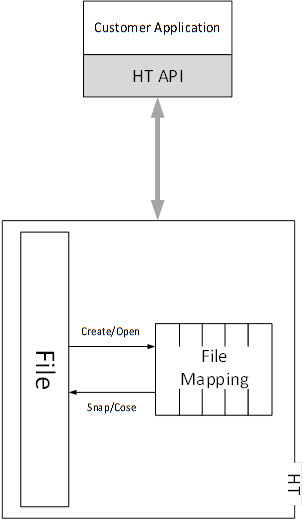
Для хранения данных в HT используется файл на диске и *образ* этого файла в оперативной памяти.

Операции с данными (**Insert, Delete, Update, Get**) выполняются **с *образом***.

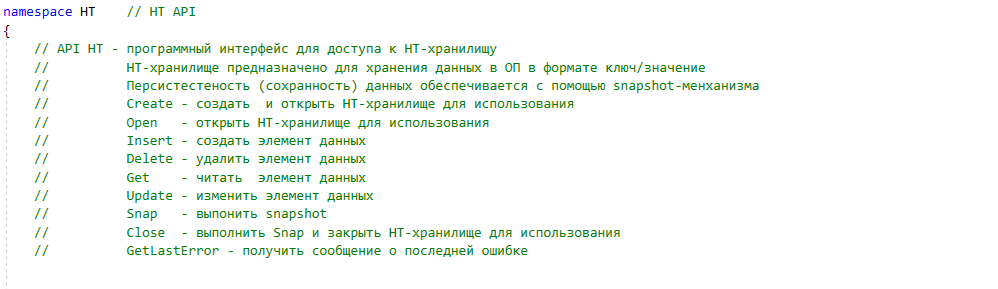
Данные образа организованы в виде хэш-таблицы с открытой адресацией.

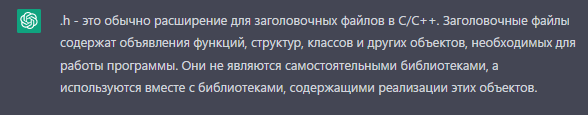
Сохранность данных осуществляется с помощью асинхронного *snapshot*- механизма (с заданным периодом асинхронное сохранение образа в файле), *snapshot* может выполняться синхронно по команде пользователя.

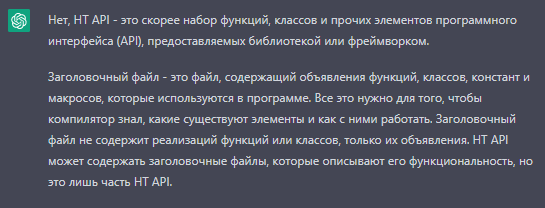
Для использования HT в приложении предназначено потокобезопасное HT API.

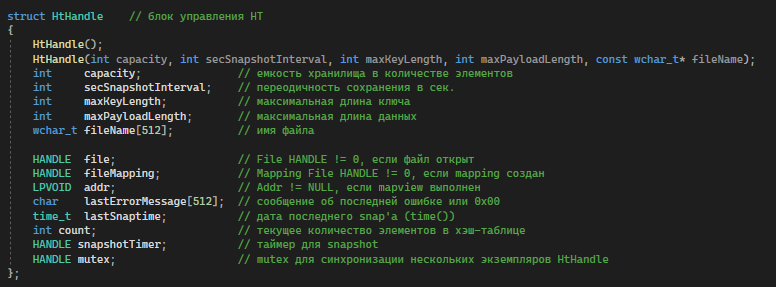


#### **HT API Ht.h**

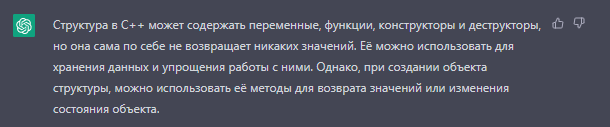


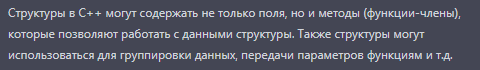




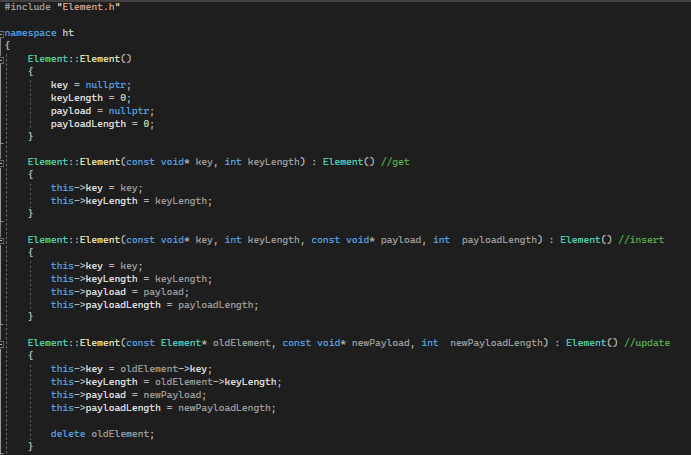








#### **Element.cpp**

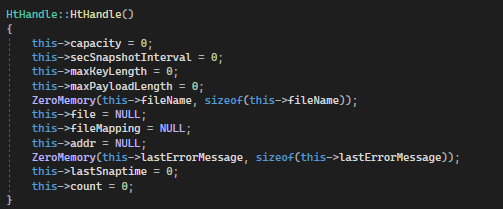


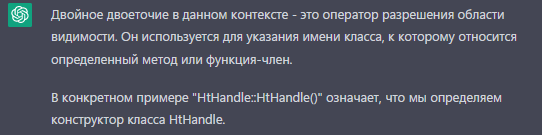
4 конструктора

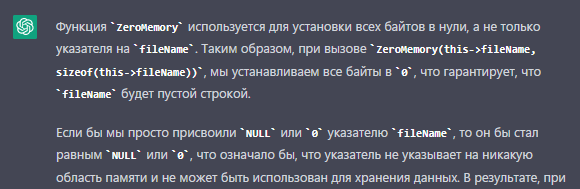


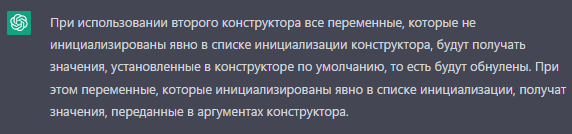
Размер 1 элемента = макс длина ключа + макс длина данных + 2 инта для хранения keyLength и payloadLength

#### **HT.cpp**

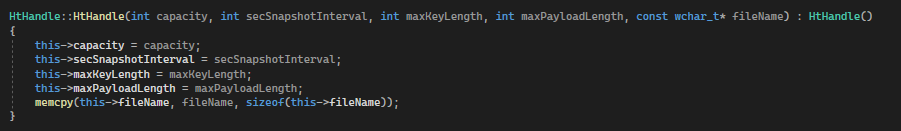


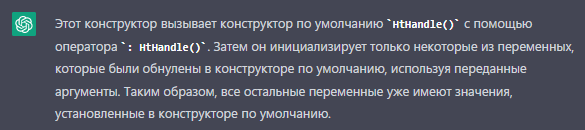




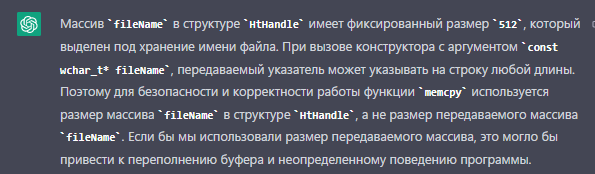


##### **HtHandle()**









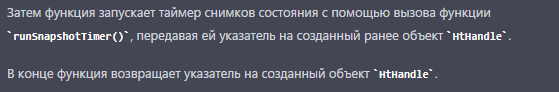
Храним саму строку, а не ее указатель, ибо при передаче в конструктор может быть Переполнение буфера

Размер поля **fileName** равен 512 \* размер **wchar\_t**

##### **create**

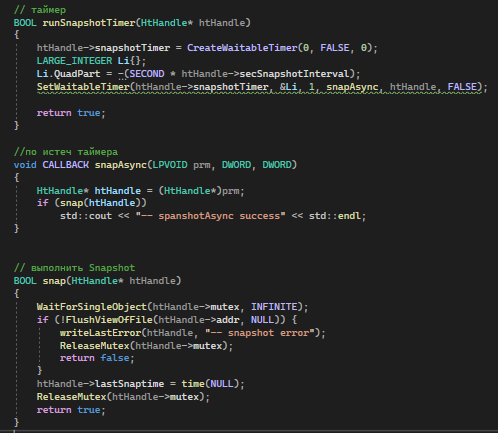


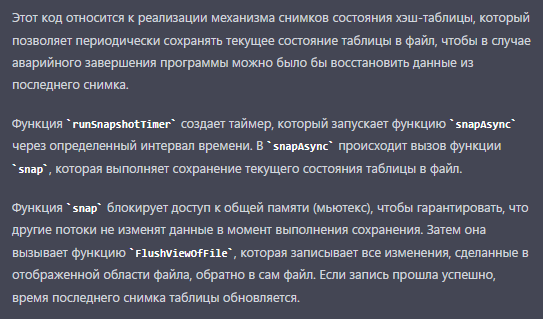
**createHt** – создает хранилище в виде хеш таблицы



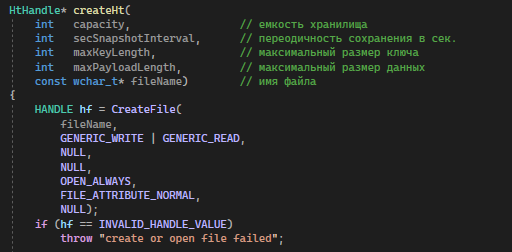
**runSnapshotTimer** является функцией, которая запускает таймер для периодического сохранения данных хранилища в файл.

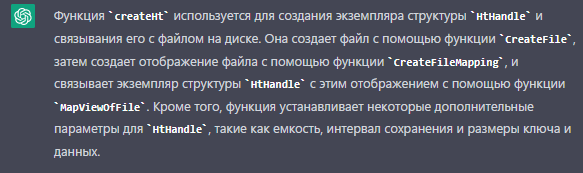
##### **Snap**

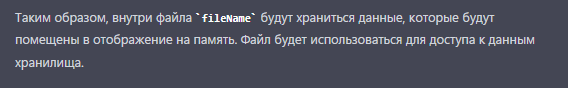




##### **CreateHt**





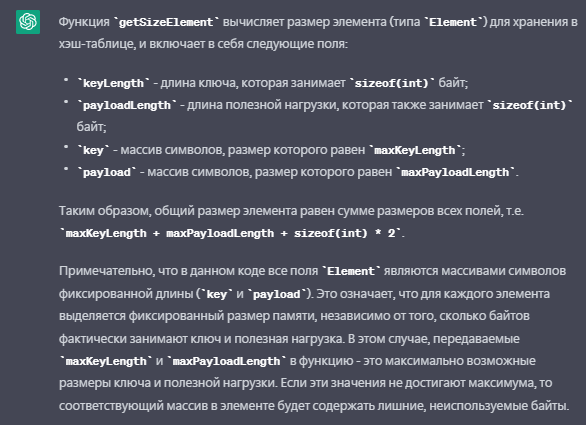


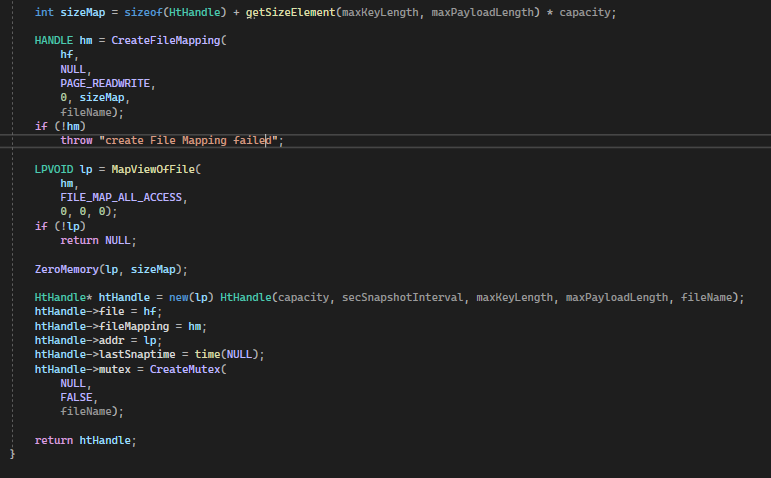
Возвращается указатель на дескриптор, потому что дескриптор в данном случае является структурой, а указатель на нее используется в качестве единого идентификатора хранилища

Использование указателя на дескриптор позволяет избежать копирования больших структур при передаче их в качестве аргументов функций и уменьшить затраты на работу с памятью. Кроме того, это позволяет удобно использовать операции с указателями, такие как сравнение и арифметические операции.

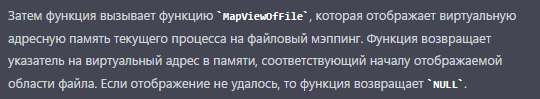
##### Размер данных для создания **createFileMapping:**

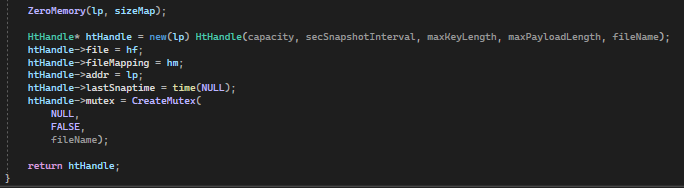






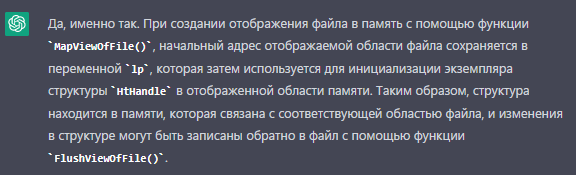




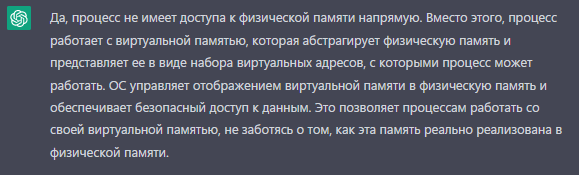


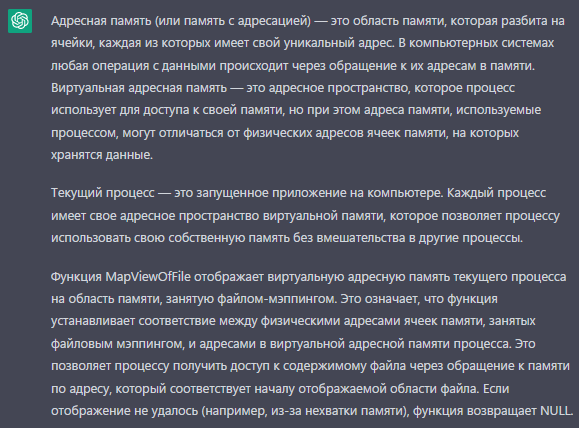
Обнуляем место в адресном пространстве на которое указывает указатель lp, создаем экземпляр структуры **new(lp)** внутри памяти адресного пространства на которое указывает lp.

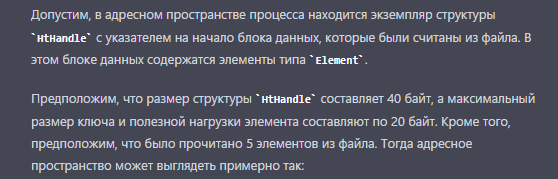
Теперь экземпляр структуры связан с файлом на диске с помощью проецирования файла в адресное пространство процесса.

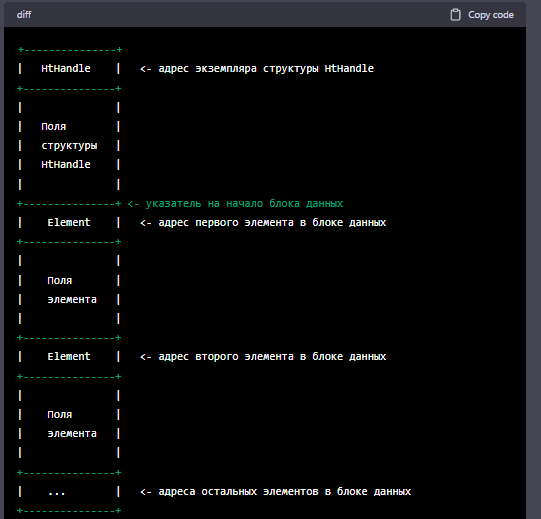


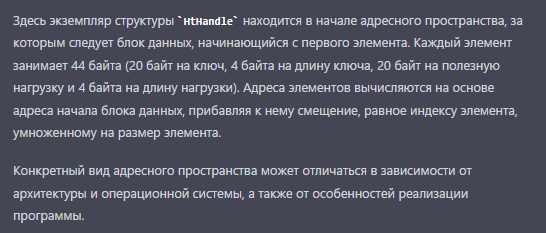
##### **===Адресное пространство процесса**







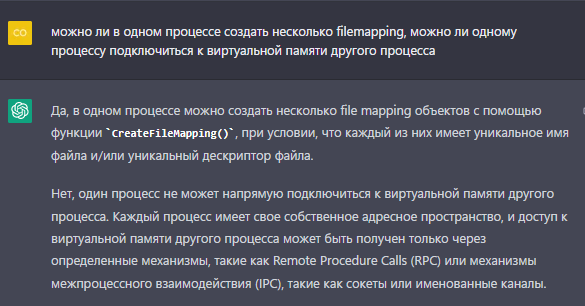




**Т.е в памяти так:**

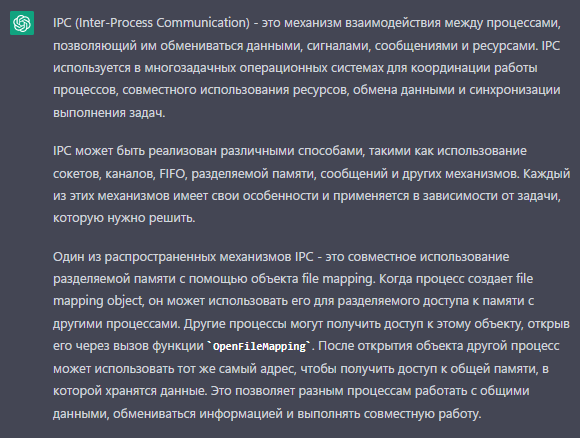
Экземпляр структуры htHandle (для доступа к полям которые содержат инфу об хеш таблице) -> элементы структуры (ключ, длина ключа(int),данные, длина данных(int)

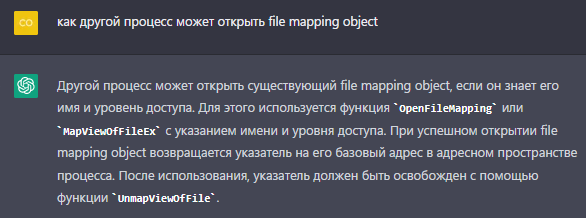
##### **===Можно ли в одном процессе создать несколько filemapping, можно ли одному процессу подключиться к виртуальной памяти другого процесса** (один из вопросов)



У каждого уникальный дескриптор должен быть

##### **===Межпроцессное взаимодействие (один из вопросов)**



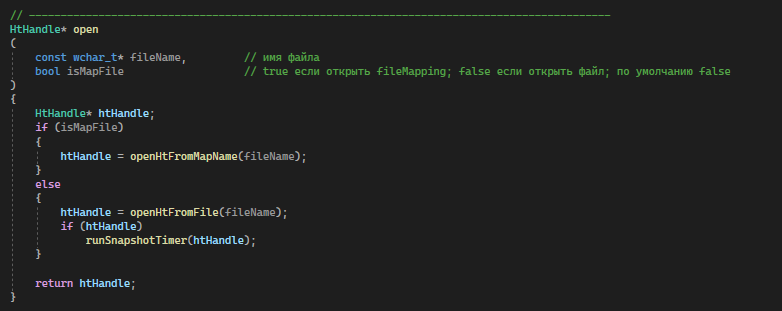


1 процесс создает FileMapping object именованный, 2 процесс подключается по имени

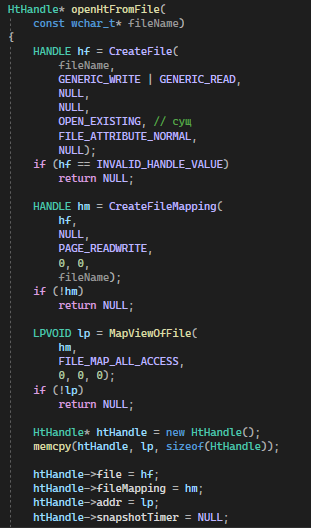
С помощью **CreateFileMapping**(указываем имя) либо с помощью **OpenFileMapping**(указываем имя)

Таким образом процессы могут реализовать **IPC** и обмениваться данными между собой.

##### **Open**



###### **Открытие из существующего файла(либо его создание):**



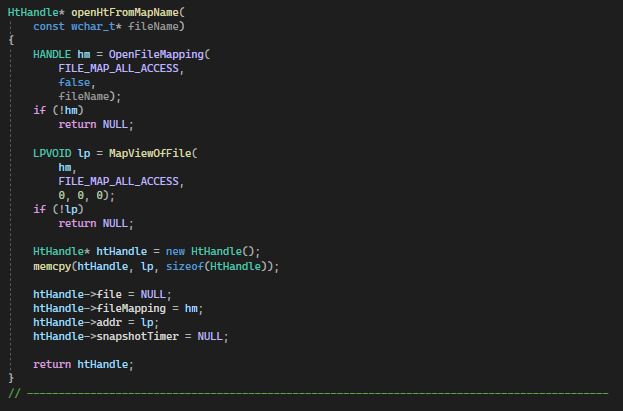
1. Open existing – открыть сущ (без создания нового)

2. File Mapping – файловая проекция, размером с весь открытый файл

3. MapViewOfFile – проекция в адресное пространство, размером с всю файловую проекцию

Создаем новый экземпляр htHandle в куче, копируем тот экземпляр который был создан изначально из памяти по адресу lp -> в наш новый экземпляр htHandle (функция memcpy(куда,откуда, сколько)

###### **Открытие существующего FileMapping object:**

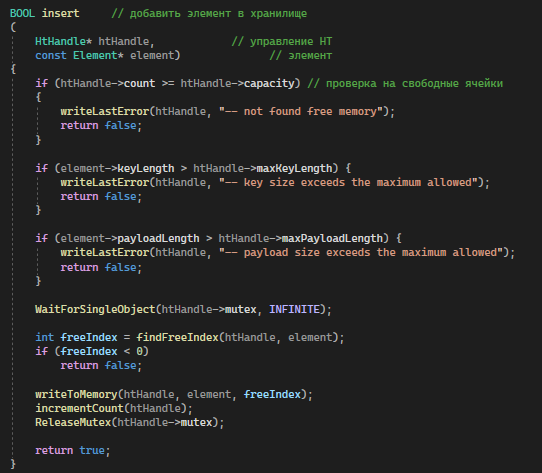


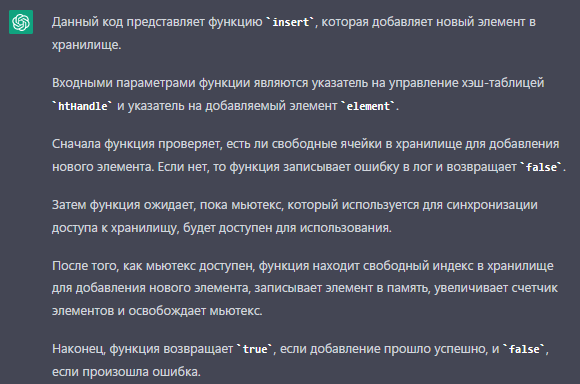
Если же у нас File mapping уже есть, то мы можем открыть его по имени и работать с теми же данными

1. OpenFileMapping + указываем имя
2. Получаем новый указатель lp который показывает новый адрес спроецированных данных из file mapping object (**т.е получили все данные что были в файловой проекции существующей!!!)**
3. Копируем память на которую указывает lp, в наш новый экземпляр htHandle (тем самым копируем все поля структуры которые находились по тому адресу) **\*через отладку F11 все подробнее можно посмотреть\***
4. Заполняем поля экземпляра структуры, при этом file у нас NULL, тк мы открываем просто File Mapping без привязки к файлу.

Теперь мы работаем с общим File Mapping object , используя OpenFileMapping с именем.

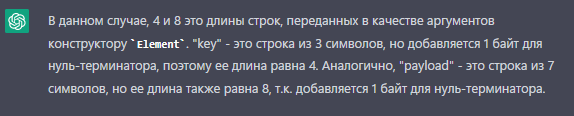
##### **Insert**

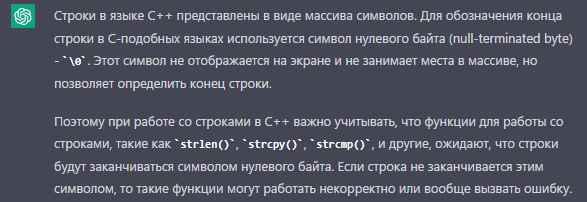






New Element() возвращает указатель





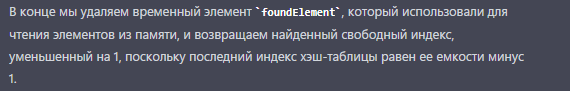


##### **FindFreeIndex**

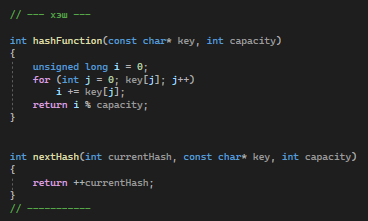


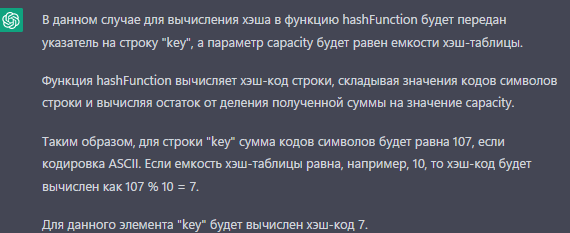
Тут происходит реализация открытой адресации, линейный перебор

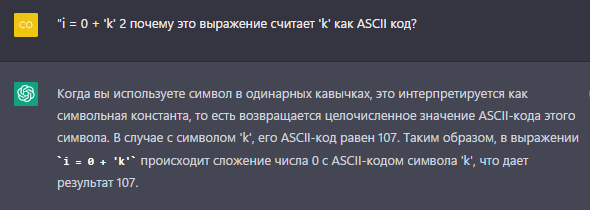




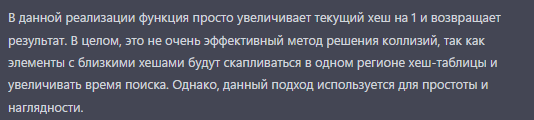
##### **Хэш функции**

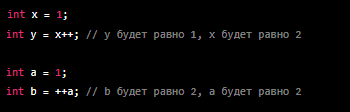






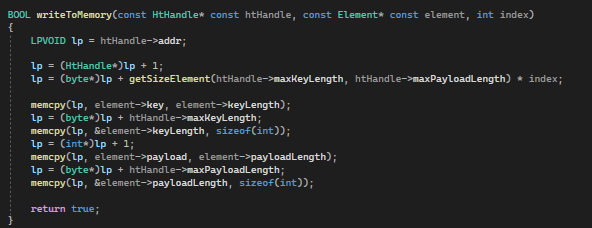




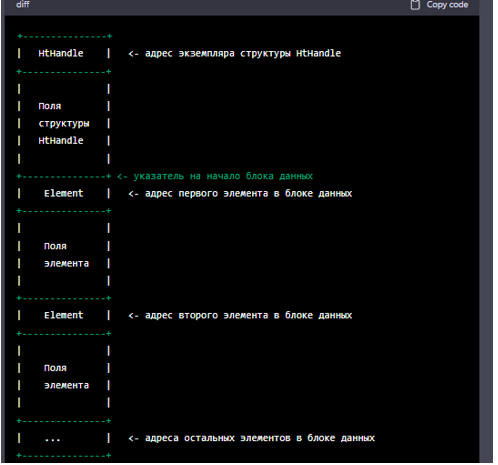


##### **writeToMemory**

заполнение памяти(поля элемента) по указателю

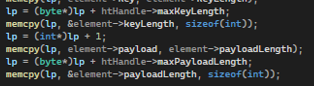


1. Получаем указатель на область памяти где хранится хэш таблица **htHandle -> addr;**
2. Увеличиваем указатель на размер структуры (HtHandle\*)lp + 1 , (тк в начале памяти хранится сама структура, потом ее элементы)



1. Теперь указатель указывает на начало блока данных (наши элементы хэш таблицы)
2. 

Переносим указатель на размер элемента\*индекс , чтобы переместиться в начало блока где содержится **поле Key элемента** под соответствующим индексом

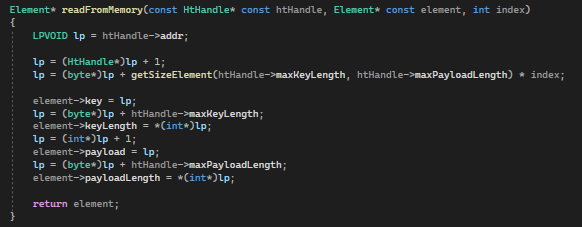
1.  заполняем этот блок памяти ключом который мы передаем в функцию
2. 

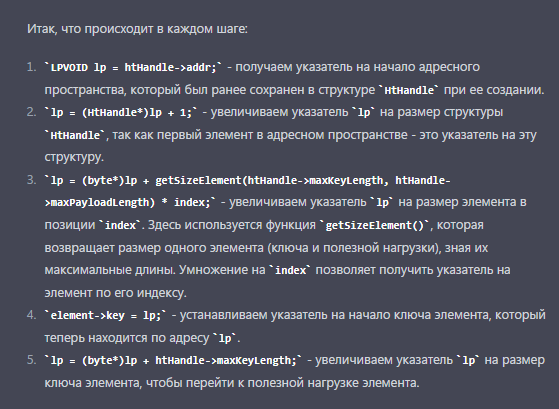
Перемещаемся к след блоку памяти добавляем **maxKeyLength** , получаем блок памяти который содержит **поле длина ключа** и записываем в этот блок данные

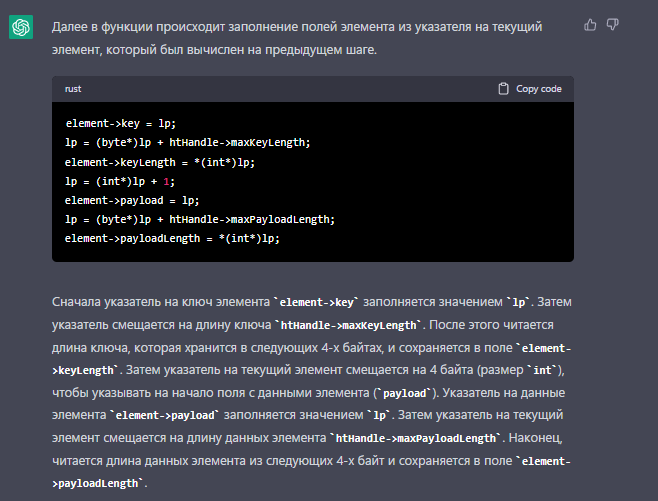
1. lp (int\*)lp + 1 – **сдвиг на 4 байта (размер инт)**
2. Получаем **поле payload данных**, записываем в память
3. Сдвигаем на размер данных и получаем **поле payloadLength длина данных**

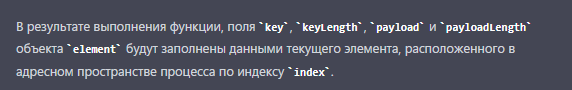
##### **readFromMemory**

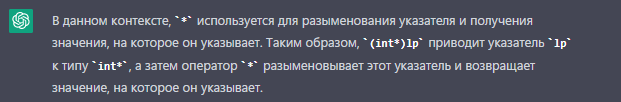
чтение памяти(полей элемента) по указателю





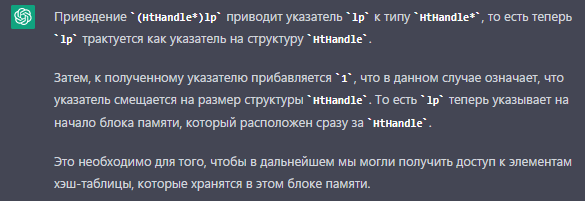




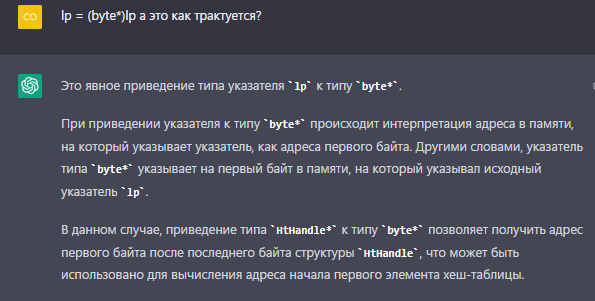


разыменование

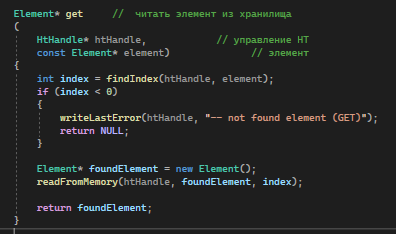


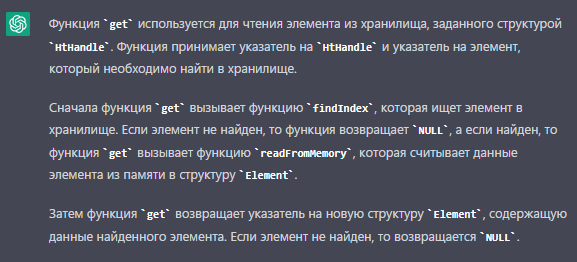




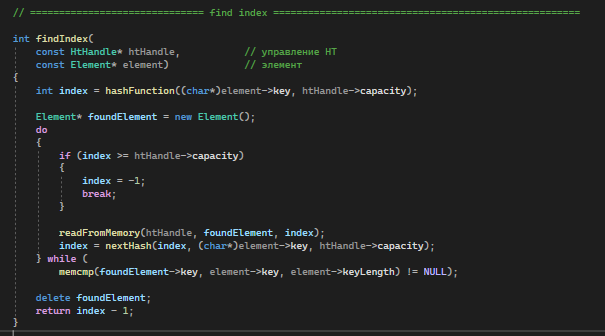


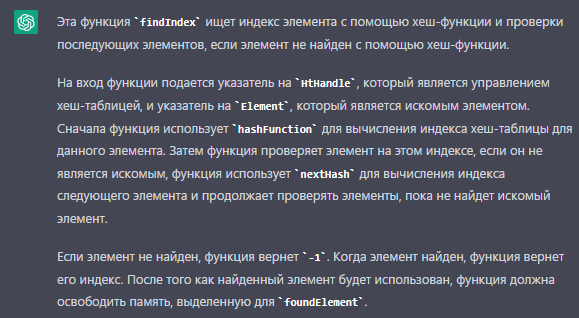
##### **Get**



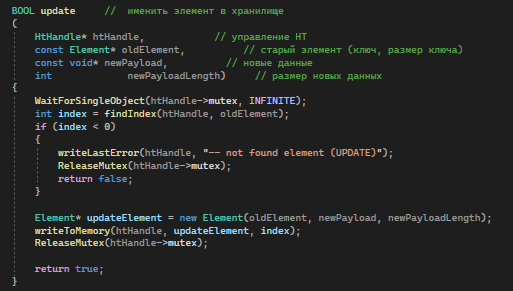


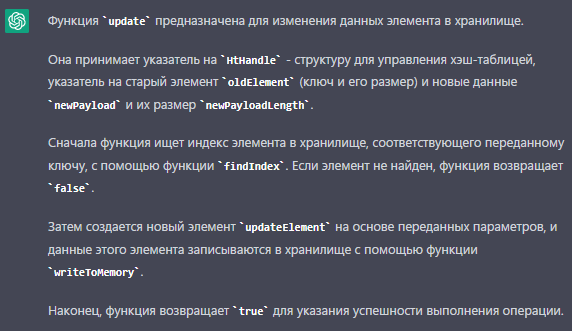
##### **FindIndex**



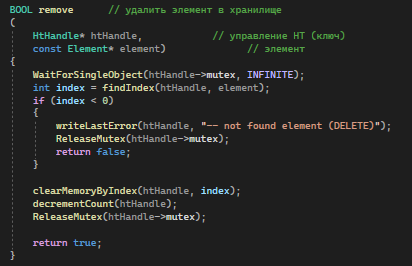


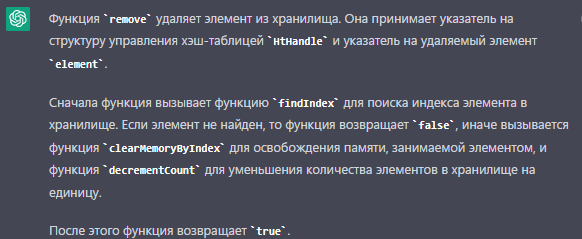
##### **Update**



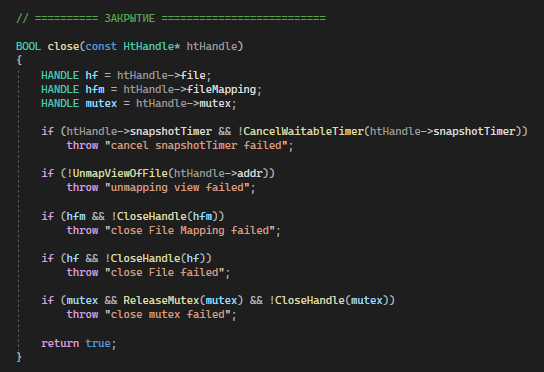


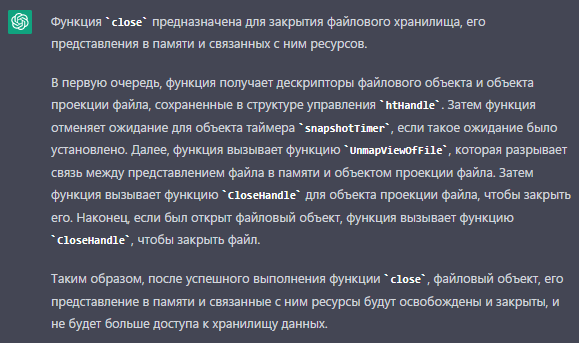
##### **Remove**





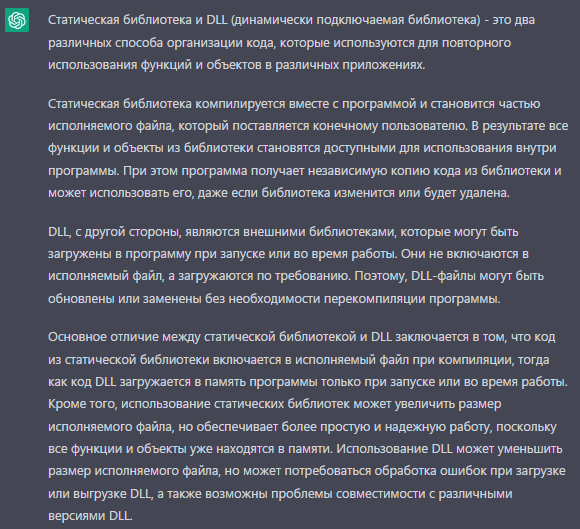
##### **Close**

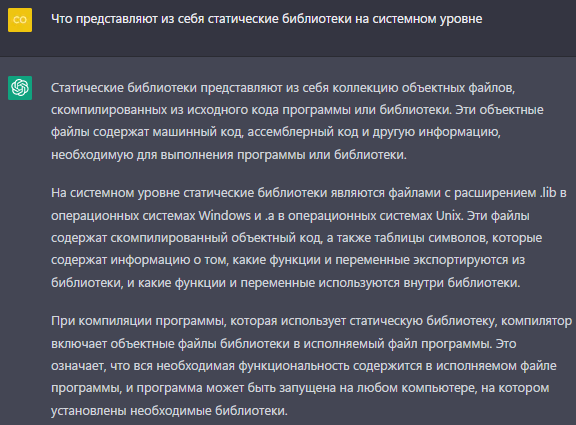


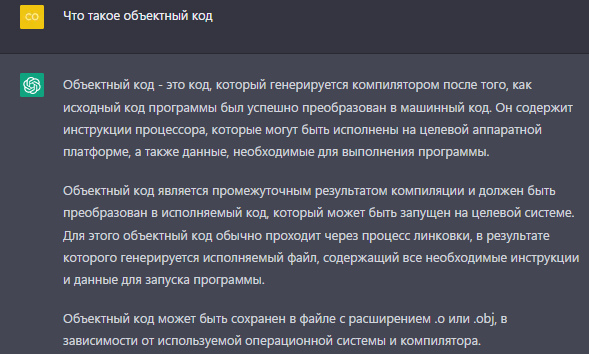


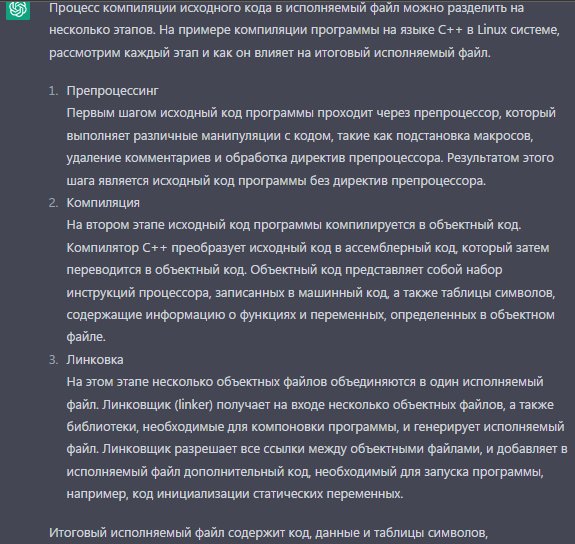
### Статическая библиотека

##### **===статические библиотеки: где хранятся, как создаются, как используется, что из себя представляет на системном уровне. отличие от dll**



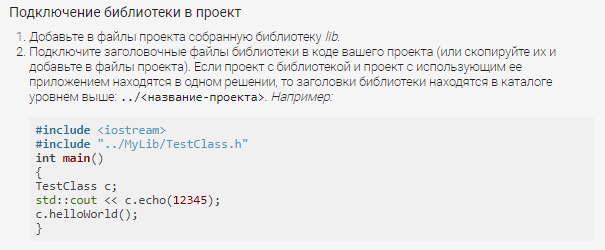






Был вопрос на каком этапе подкл статическая библиотека -> на этапе линковки (объединения в исполняемый файл)





### ===Что может спросить по лабе:

1) Что такое filemapping, как он работает, что такое адресное пространство процесса, как происходит межпроцессное взаимодействие с помощью filemapping

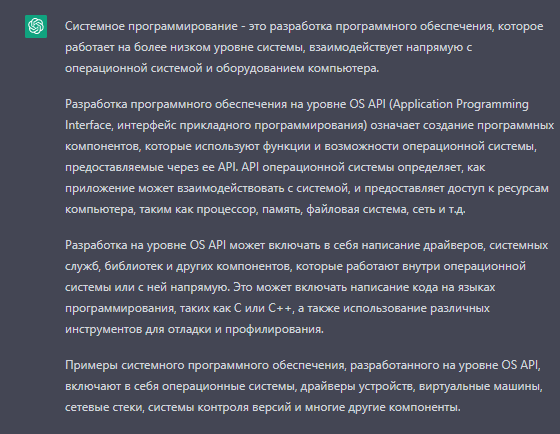
2) Что такое хэш таблица, что хэшируется, что такое коллизия, как ее решали в лабараторной работе, что такое метод открытой адресации

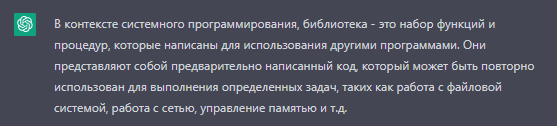
3) Что такое статическая библиотека, что из себя представляет и тд, отличие от dll

-- Про хранилище все алгоритмы create,open,insert,get и тд

### Лекция\_01\_OC\_Введение

**Системное программирование:** разработка программного обеспечения на уровне **OS API**.

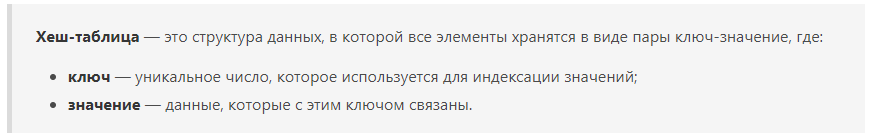


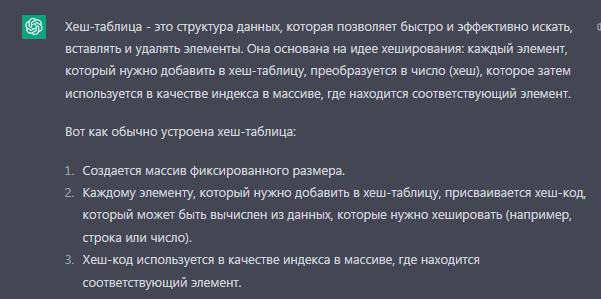


1. **Прикладное и системное программирование:** пользователем прикладного программного обеспечения является конечный пользователь (не обязательно человек), системное программное обеспечение является **вспомогательным средством для разработки** прикладного программного обеспечения (библиотеки функций, фреймворки, компиляторы, …).

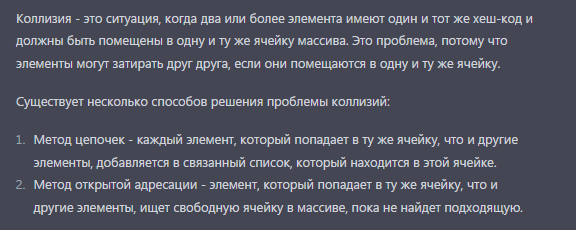
**Цель курса:** получить **навыки разработки API для прикладных программных** систем **на базе OS API**.

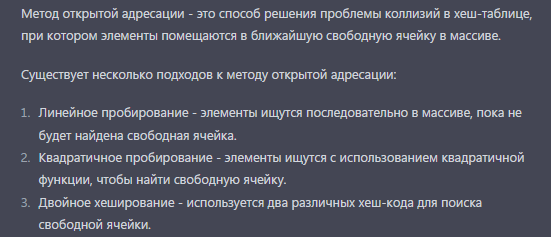
### ===Хэш таблица (один из вопросов)





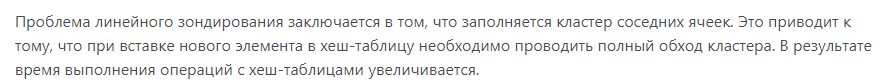
#### **Коллизия:**

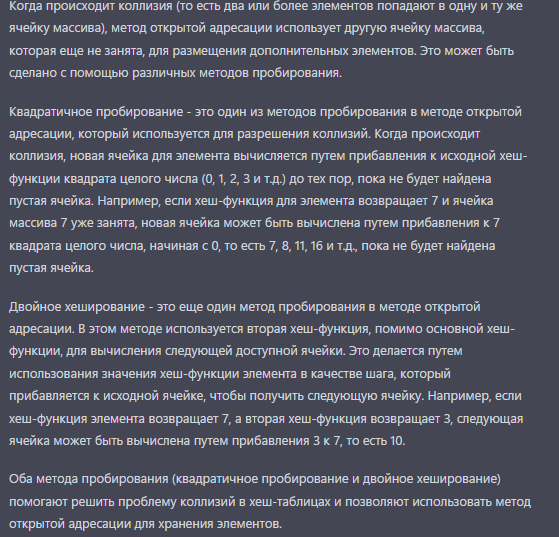




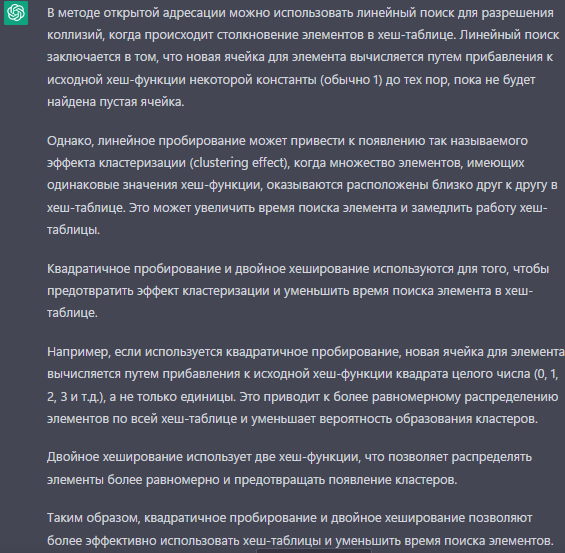
###### **Линейное, квадратичное, двойное хеширование:**

Линейное пробирование – элементы ищутся последовательно пока не будут найдены.

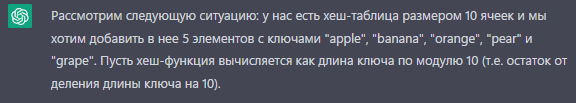


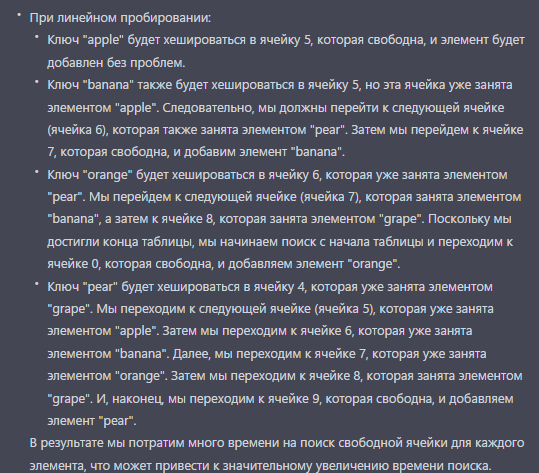


###### **Зачем вообще линейное квадратичное и двойное:**

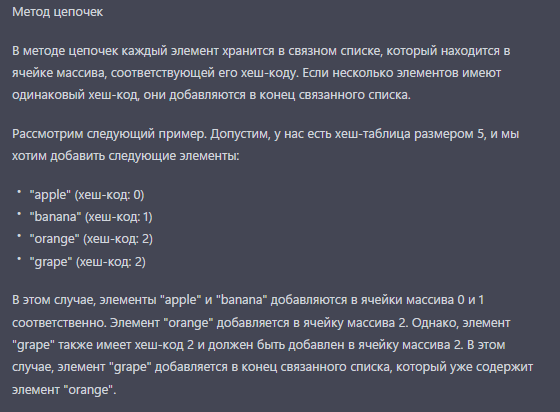


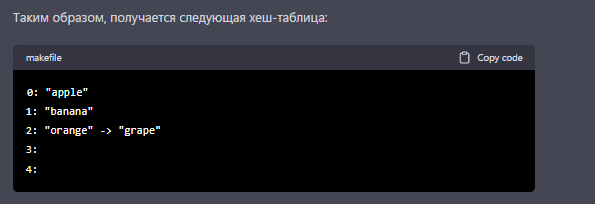
Почему время поиска увеличится при линейном:





##### **Метод цепочек:**





###### **Связной список:**

