**10 лаба HT, File Mapping, lib**

**---Что может спросить по лабе:**

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

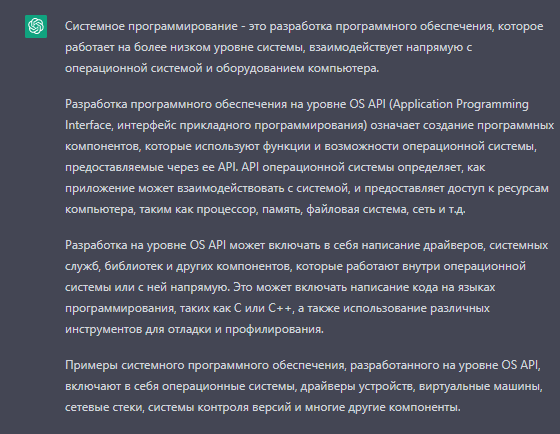
2) Что такое хэш таблица, что хэшируется, что такое коллизия, как ее решали в лабараторной работе, что такое метод открытой адресации

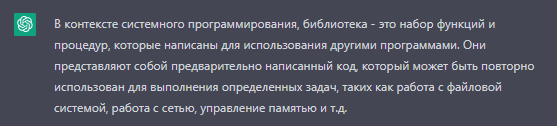
3) Что такое статическая библиотека, что из себя представляет и тд, отличие от dll

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

**Лекция\_01\_OC\_Введение**

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

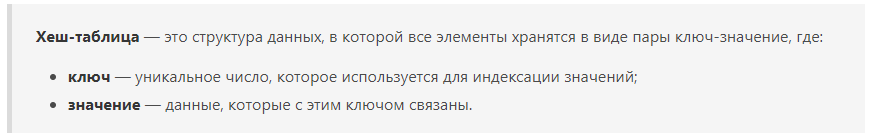


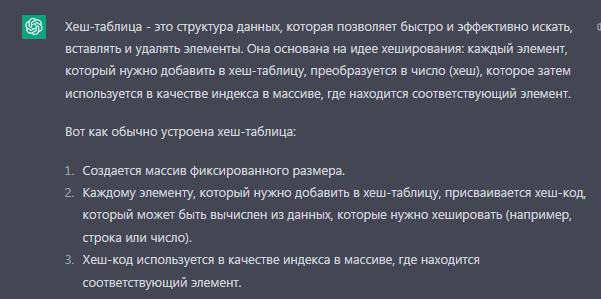


1. **Прикладное и системное программирование:** пользователем прикладного программного обеспечения является конечный пользователь (не обязательно человек), системное программное обеспечение является **вспомогательным средством для разработки** прикладного программного обеспечения (библиотеки функций,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

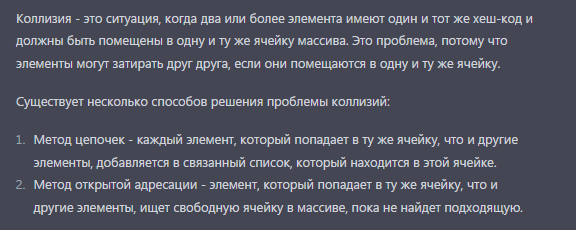
**Цель курса:** получить **навыки разработки API для прикладных программных** систем **на базе OS API**.

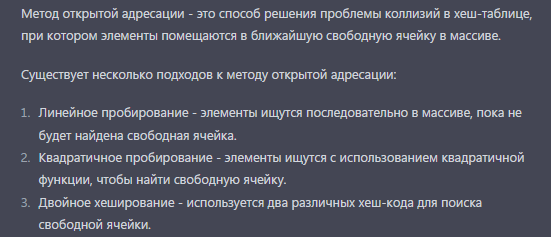
**=Хэш таблица (один из вопросов)**





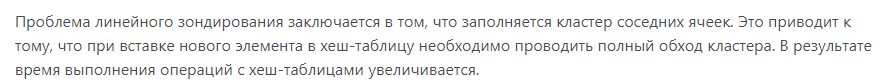
***Коллизия:***

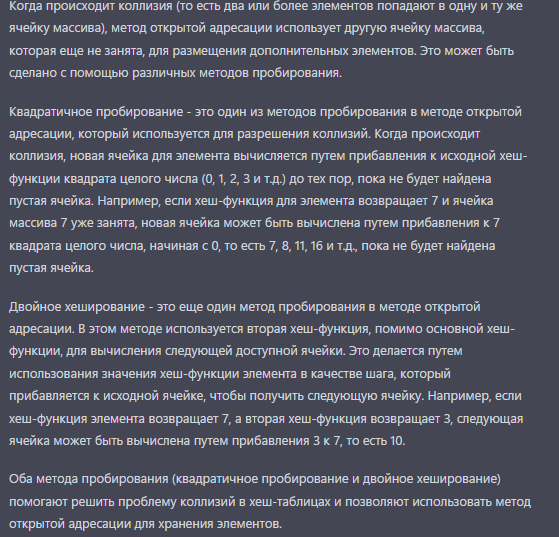




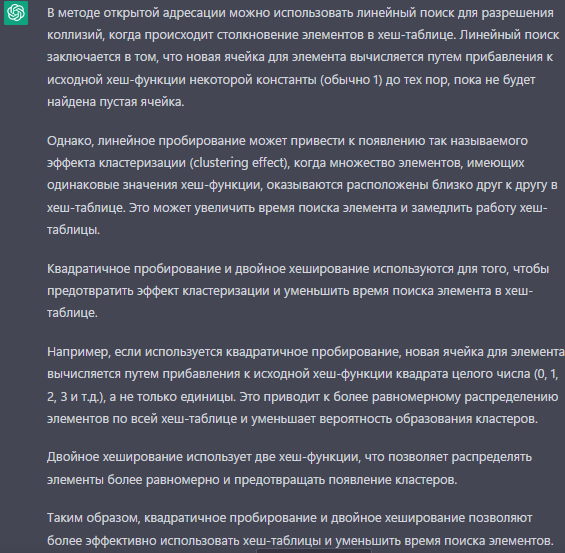
**Линейное, квадратичное, двойное хеширование:**

Линейное пробирование – элементы ищутся последовательно пока не будут найдены.

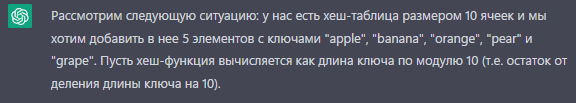


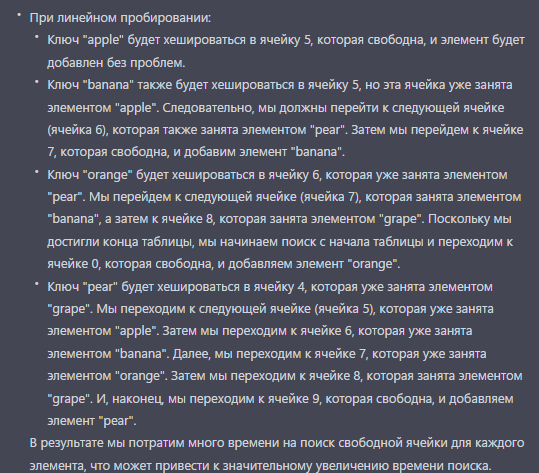


**Зачем вообще линейное квадратичное и двойное:**

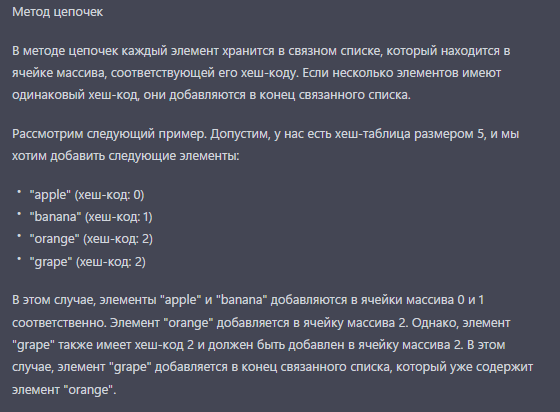


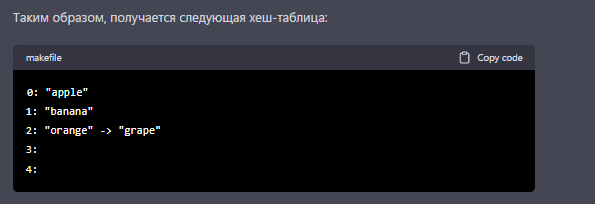
Почему время поиска увеличится при линейном:



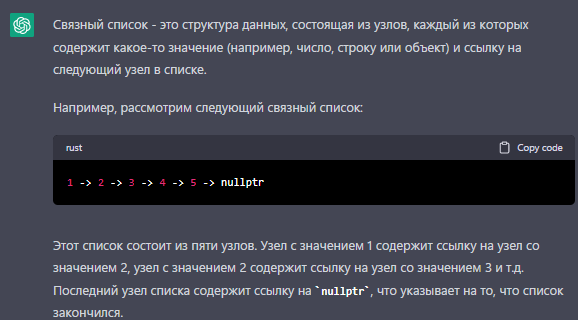


**Метод цепочек:**



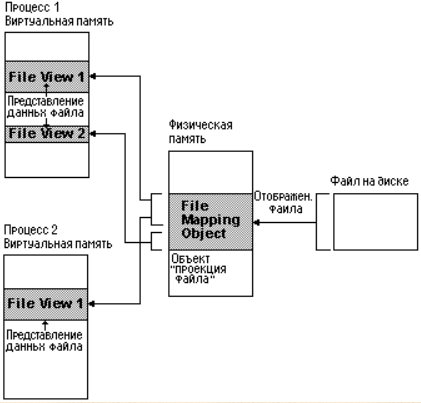


**Связной список:**



**=File Mapping (один из вопросов)**

[**https://www.youtube.com/watch?v=m6YaGKyzkak**](https://www.youtube.com/watch?v=m6YaGKyzkak)

****

**Файловые проекции: что из себя представляют и для чего это:**

**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). Вы сможете работать с данными файла обращаясь по этому указателю.**

**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.**

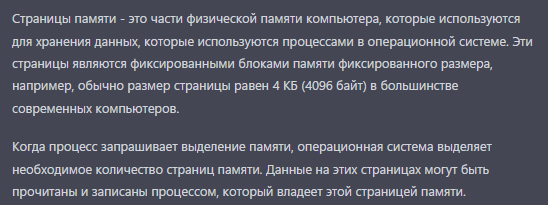
**Есть файл на диске, файловая проекция это File mapping.**

**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**

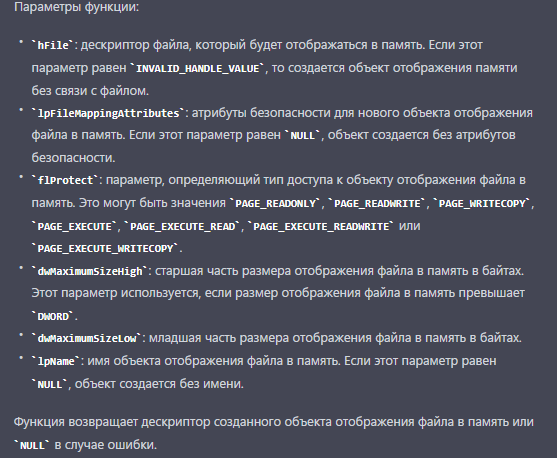
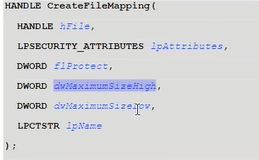
**Оперирование этими объектами (file maping, file view) кратно страницам памяти. Потому что занимаются целые страницы и проецируются в виртуальное адресно пространство процесса.**



**Как с этим всем работать:**

1. **Создать файл CreateFile**
2. **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**

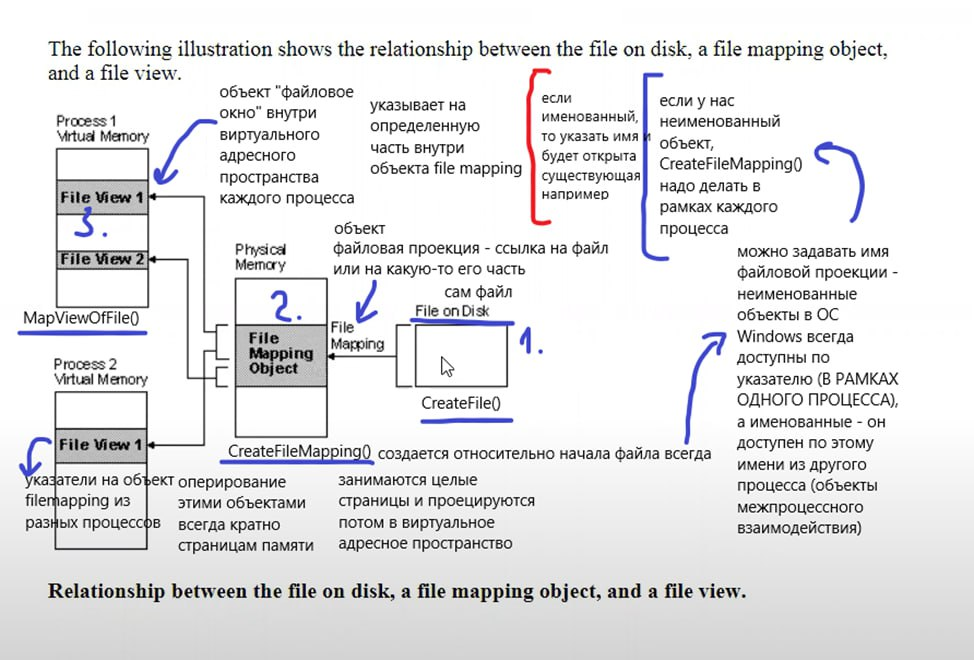
**Объекты ОС которые могут быть именованы и не именованы:**

**Неименованные объекты в ОС всегда доступны только по указателю (только в рамках одного процесса), если объект именованный он может быть доступен по имени из другого процесса (как правило это объекты для межпроцессного взаимодействия).**

**Можно создать файловую проекцию и открыть ее из разных процессов, работать с одним и тем же набором данных и обмениваясь ими (если на swop file создать то мы работаем с данными в памяти вообще а не с файлом).**

**Если мы будем осуществлять межпроцессное взаимодействие, то указываем имя файловой проекции !!**

**если не нужно то просто передаем NULL**

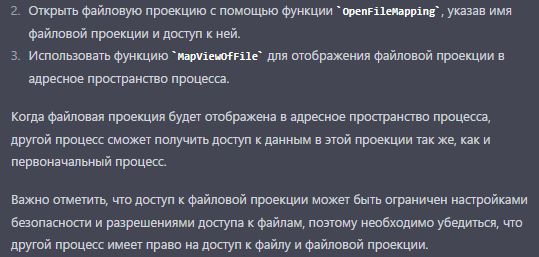


**После того как мы получили HANDLE File mapping, у нас становится доступным создание файловых окон которые будут указывать на эту файловую проекцию из разных процессов.**

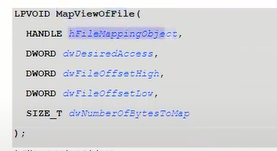
**Так как у нас есть HANDLE , то это означает, что создать файловую проекцию нужно будет В КАЖДОМ ПРОЦЕССЕ ОТДЕЛЬНО, если при создании мы укажем имя существующей проекции, она будет просто открыта, тк она уже существует и новая создаваться не будет.**

**Функция CreateFileMapping относится к каждому процессу отдельно. Если в одном процесса создали, то в другом мы тоже должны ее вызывать, если хотим получить доступ к той же файловой проекции, НО С ПЕРЕДАЧЕЙ УЖЕ СУЩЕСТВУЮЩЕГО ИМЕНИ (если именована, если нет то создаем новую).**

**После того как файловая проекция создана одним из процессов у нас появляется возможность получить к ней доступ по имени. Мы можем создавать файловые окна.**

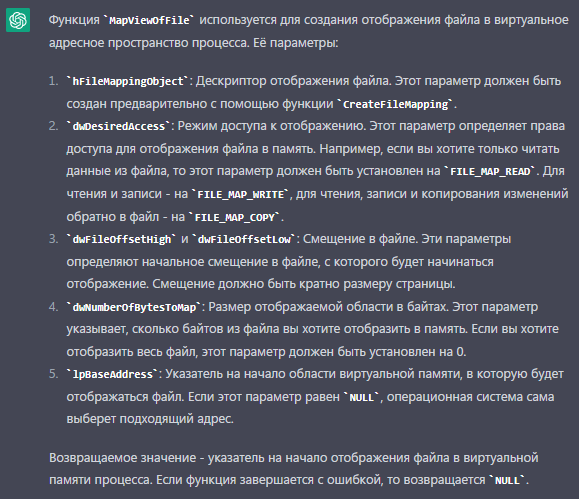


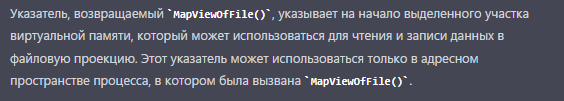
**Файловое окно создается с помощью функции MapViewOfFile**



**Дескриптор на файловую проекцию, параметры доступа(открывалась только для чтения а вы запрашиваете запись – будет ошибка), начало смещения относительно файловой проекции в 2-х значениях, размер в байтах окна которое мы хотим получить**

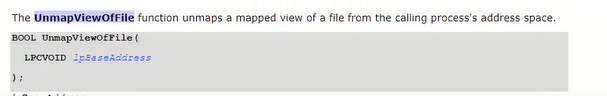
**Размер кратен страницам памяти**





**После того как мы получили файловое окно, нам возвращает LPVOID указатель, возвращает УКАЗАТЕЛЬ НА ПАМЯТЬ, то есть у нас этот файл через окно спроецирован в виртуальное адресное пространство, это обычный указатель на память (в адресном пространстве),**

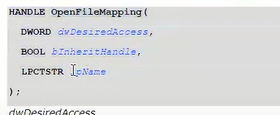
**Если FileView больше не нужна:**



**Указатель на начало памяти в адресном пространстве**

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

**Открыть существующую файловую проекцию:**

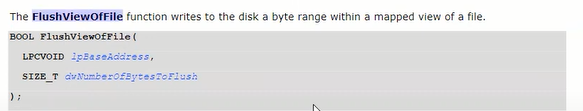


**Передаем имя существующего меппинга**

**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.**

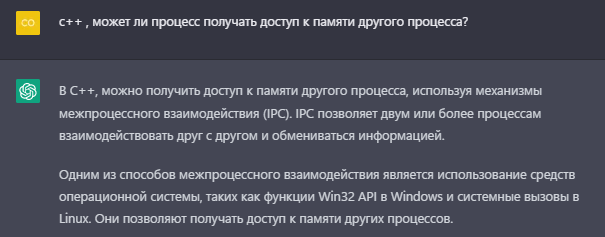
***Итог:***

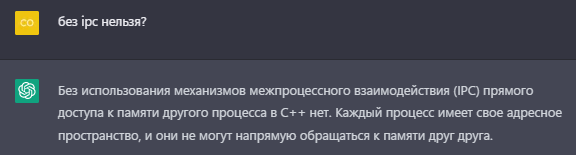
**Для того чтобы работать через файловую проекцию, нужно создать ее и споецировать в виртуальное адресное пространство процесса, где мы получим указатель на память, дальше мы работаем с данными как с памятью, *обращаемся как к массиву и тд*.**

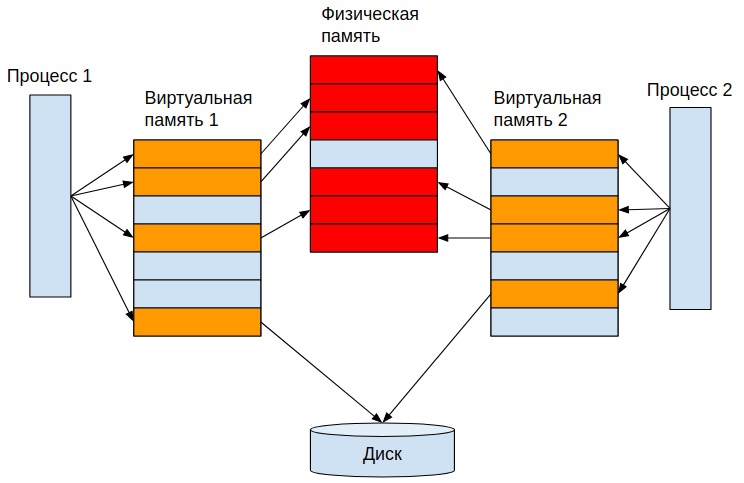
**Где применяется: обмен данными между процессами, через обычные файлы не вариант тк мы будем реально обращаться к файлам на жестком диске, а через файловую проекцию мы обращаемся через память и обмен данными осуществляется В РАЗЫ БЫСТРЕЕ.**

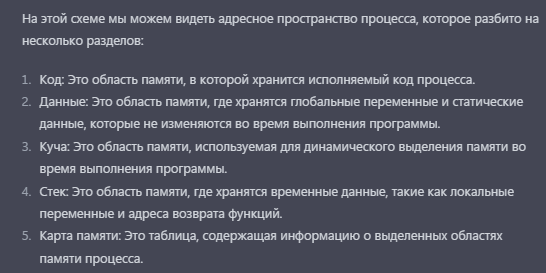
Это очень помогает при работе с огромными файлами, такими как файлы базы данных, поскольку не весь файл должен находиться в физической памяти.

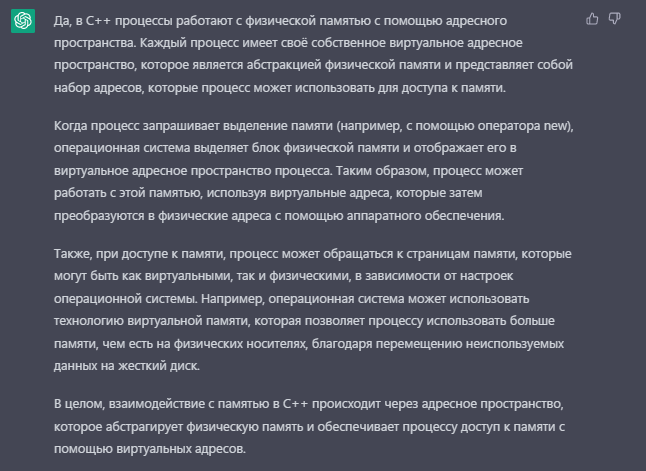
=====================================================================



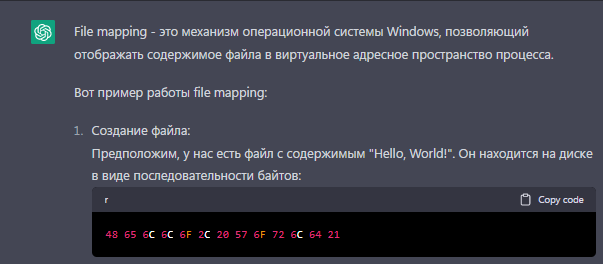


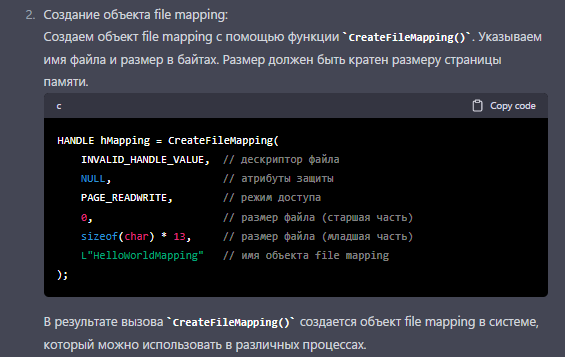


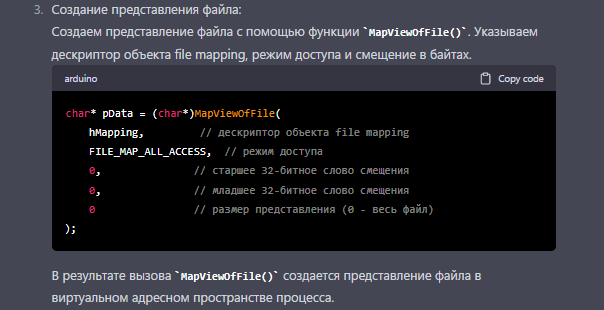


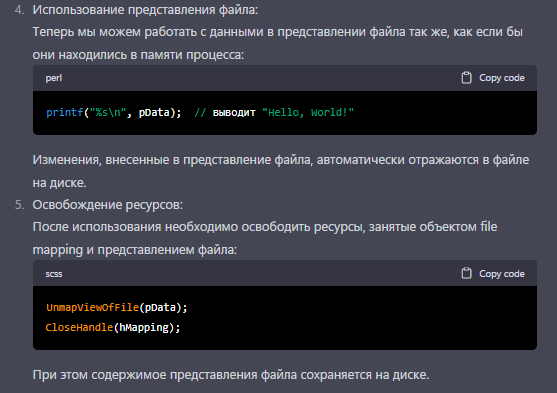


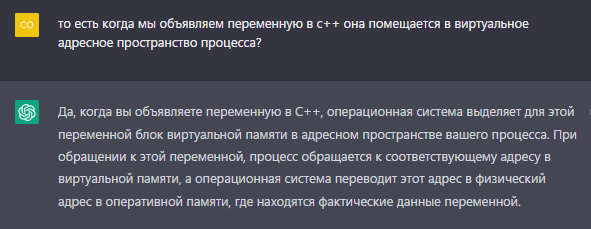
**Более просто про FileMapping**









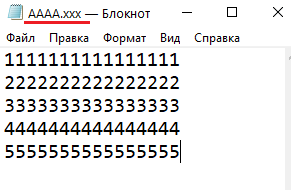


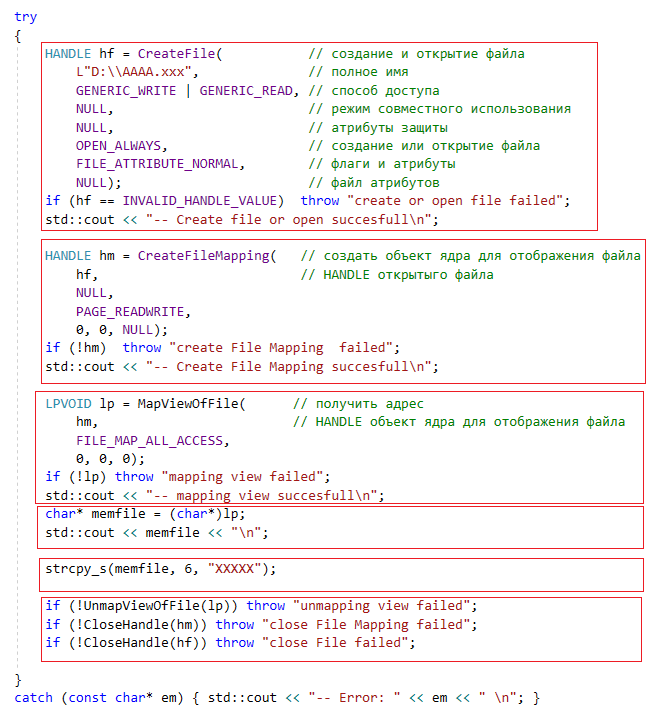
**Лекция\_09\_File\_Mapping\_V2**

**File Mapping**: отображение файла в **виртуальную память процесса**;

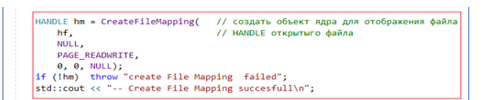
*отображенный файл называется* ***File View*** *(представлением файла).*

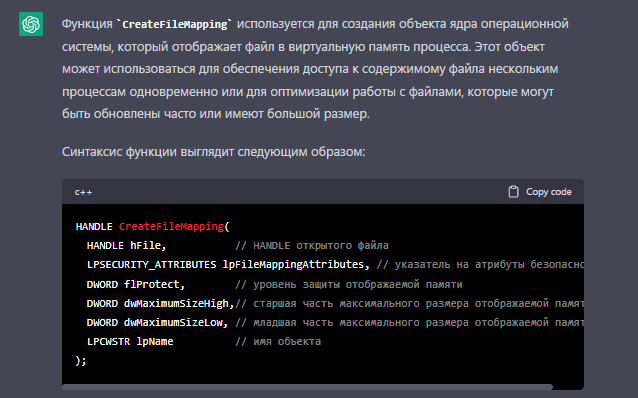
***OS: Windows: CreateFileMapping, MapViewOfFile***

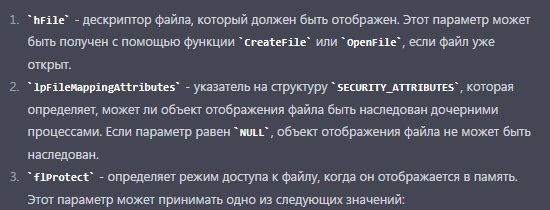
****

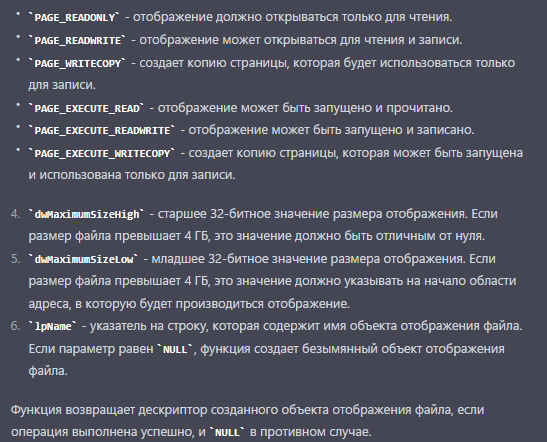
****

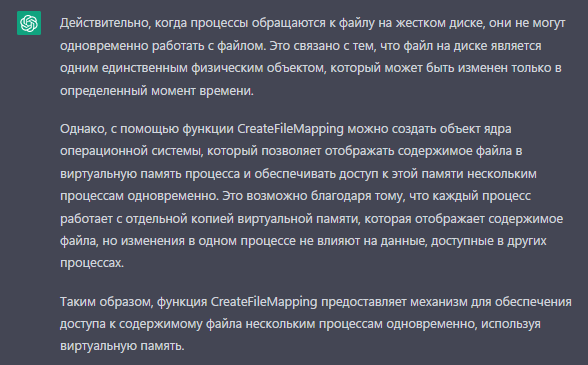
**CreateFileMapping**

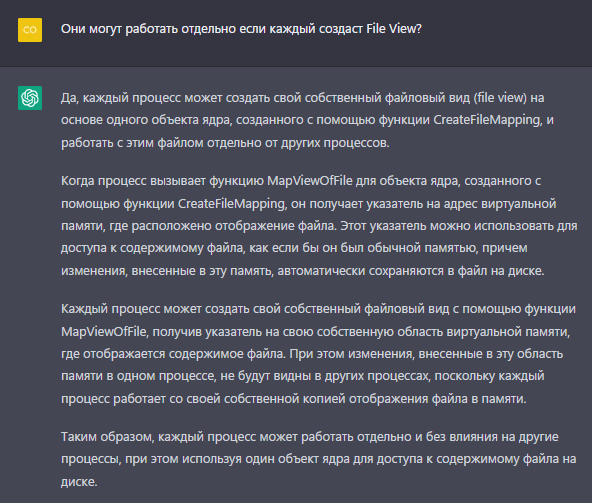


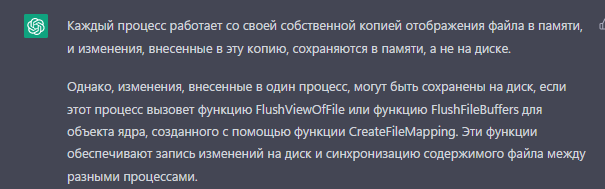




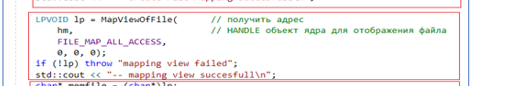


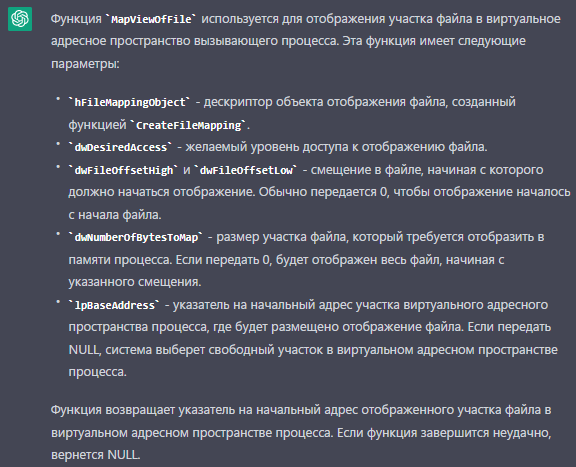


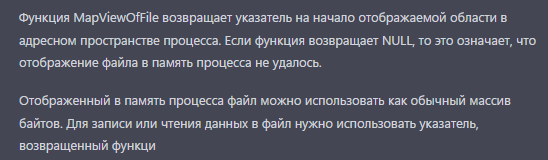




**MapViewOfFile**

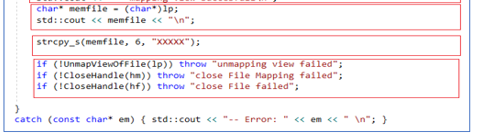


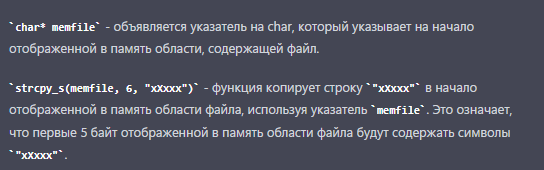


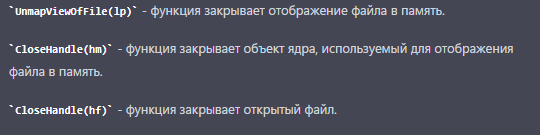


**Для записи и чтения используем указатель**

**Вывод файла и освобождение**





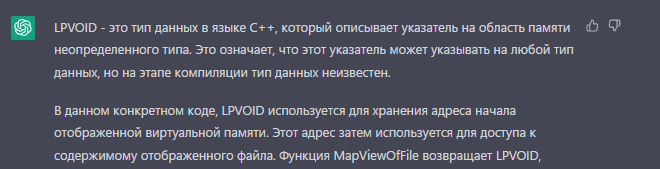


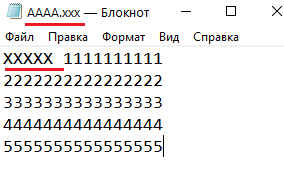
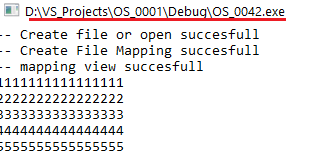
**Как я понял из-за unmapviewoffile происх запись в файл?**

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

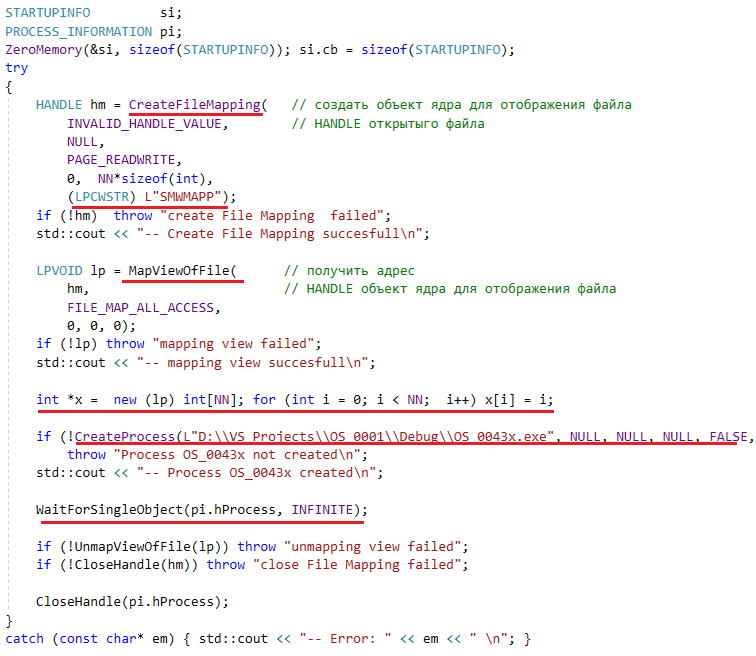


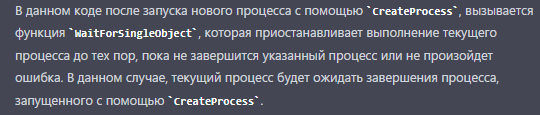
**LPVOID**



****

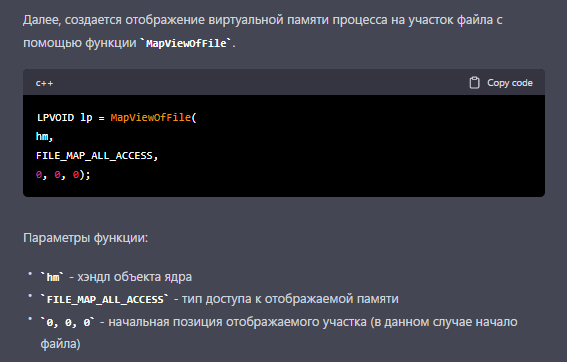
***OS: INVALID\_HANDLE\_VALUE , NAME FileMapping***

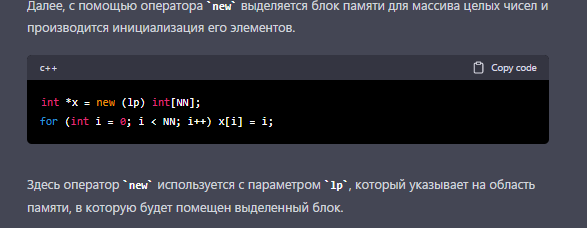
****

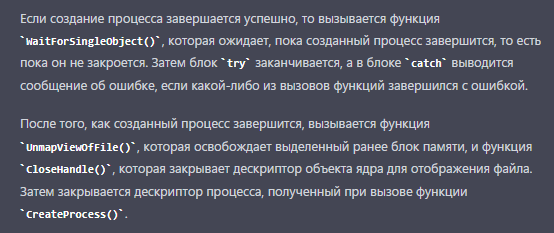


**После заверш второго идет овсвобождение viewoffile первого и closehandle**

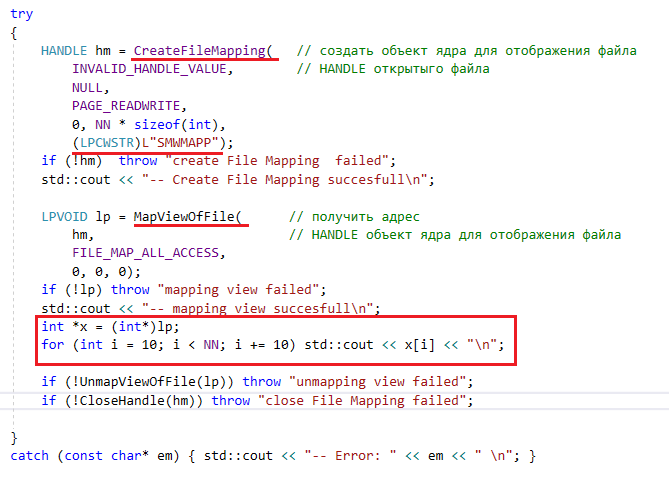


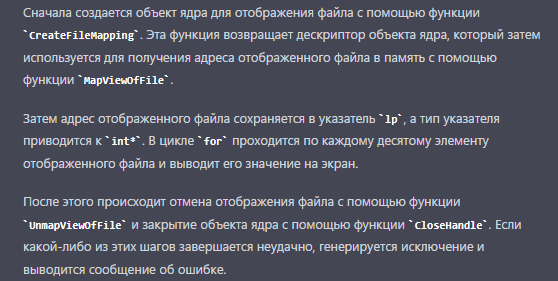


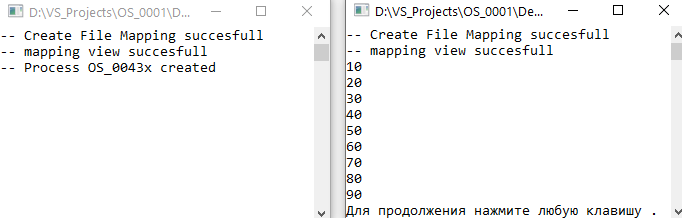




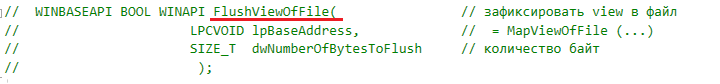
**Из другого процесса:**

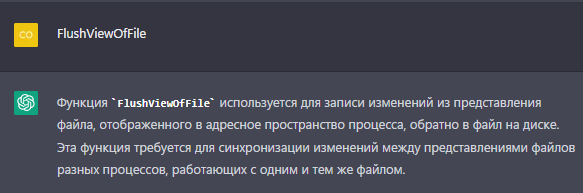
****



****

***OS:FlushViewOfFile***

****



**Из адресного пространства -> файл**

**Метода + код лабы**

***HT-хранилище (HT):***

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

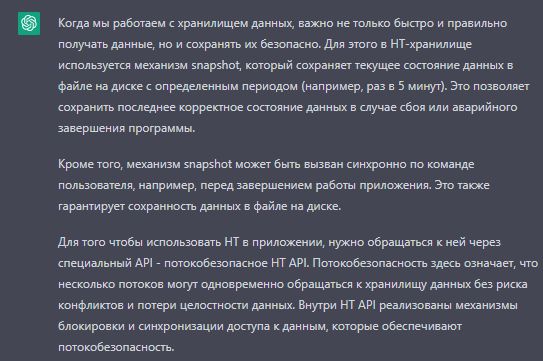
Для хранения данных в HT используется файл на диске и *образ* этого файла в оперативной памяти.

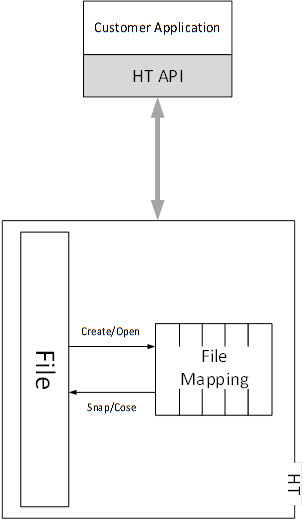
Операции с данными (**Insert, Delete, Update, Get**) выполняются **с *образом***.

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

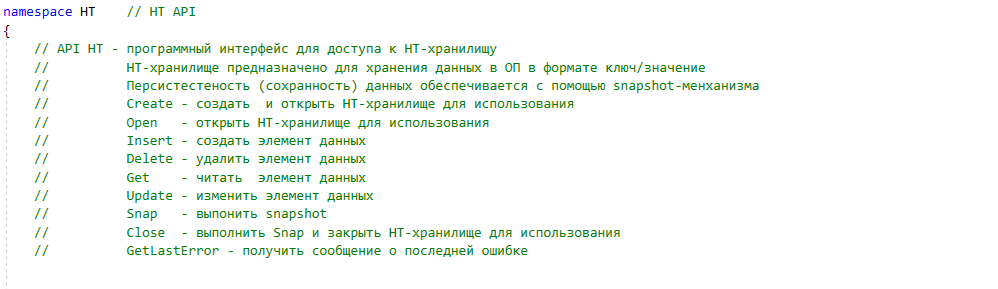
Сохранность данных осуществляется с помощью асинхронного *snapshot*- механизма (с заданным периодом асинхронное сохранение образа в файле), *snapshot* может выполняться синхронно по команде пользователя.

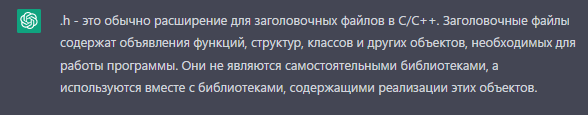
Для использования HT в приложении предназначено потокобезопасное HT API.

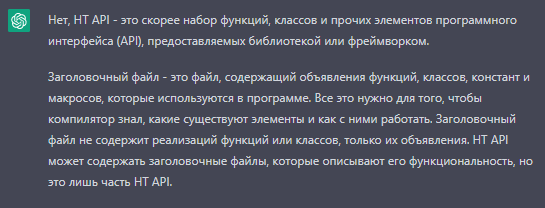


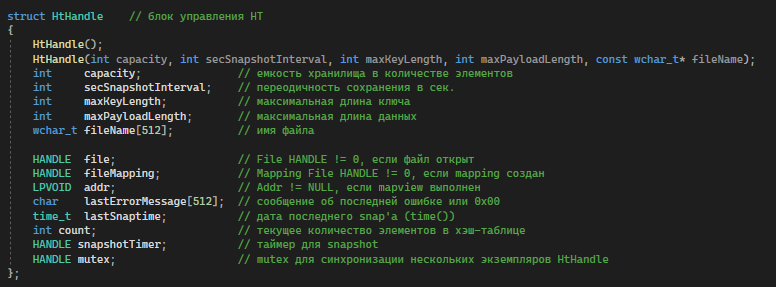


***HT API* Ht.h**

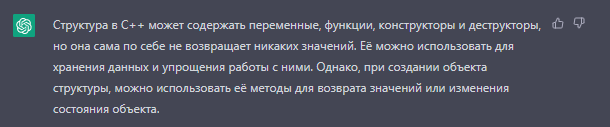


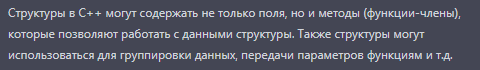




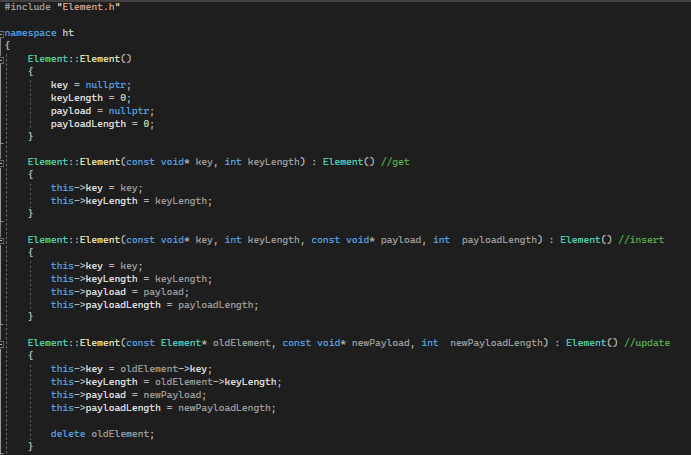








**Element.cpp**

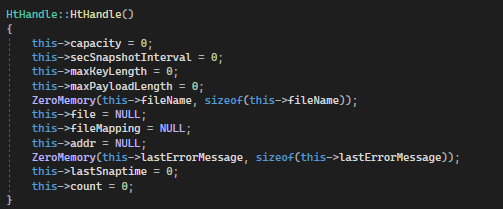


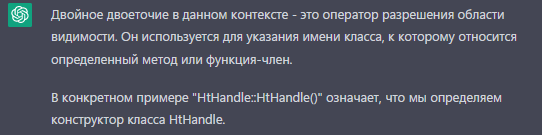
4 конструктора

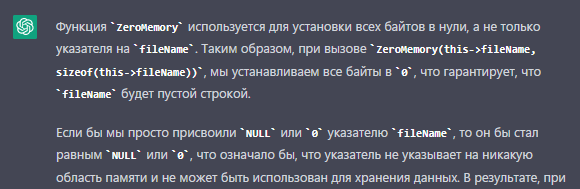


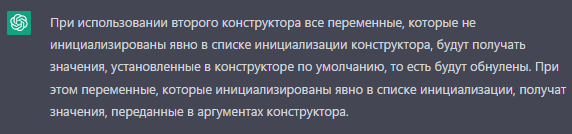
Размер 1 элемента = макс длина ключа + макс длина данных + 2 инта для хранения keyLength и payloadLength

**HT.cpp**

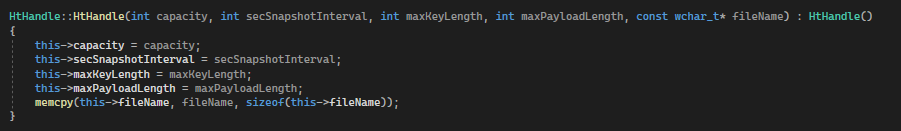


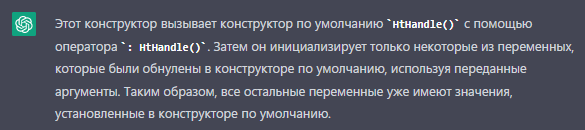




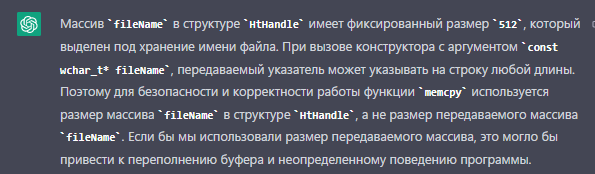


**HtHandle()**









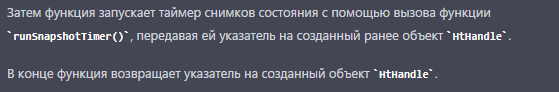
Храним саму строку, а не ее указатель, ибо при передаче в конструктор может быть Переполнение буфера

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

**create**

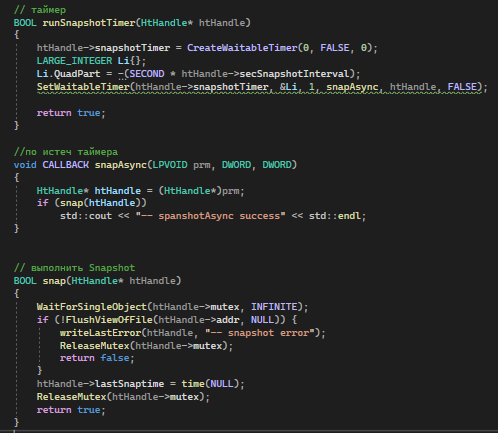


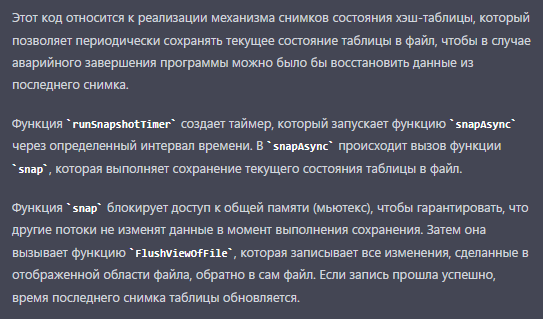
**createHt** – создает хранилище в виде хеш таблицы



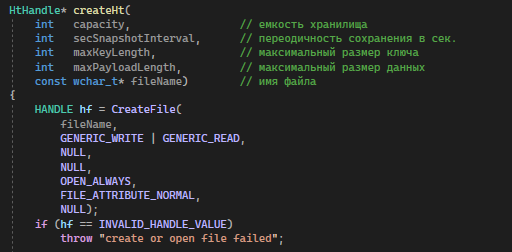
**runSnapshotTimer** является функцией, которая запускает таймер для периодического сохранения данных хранилища в файл.

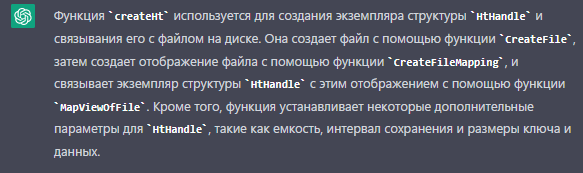
**Snap**

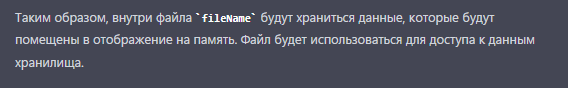




**CreateHt**





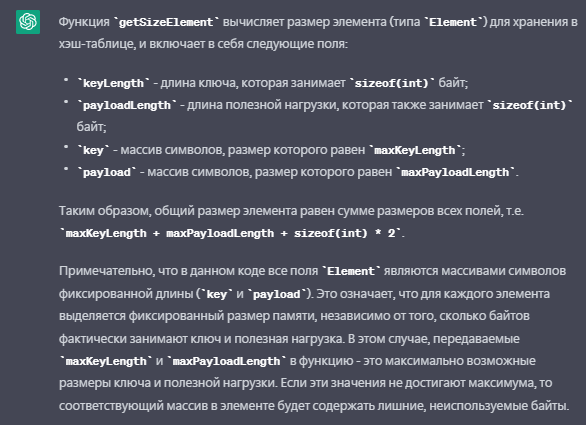


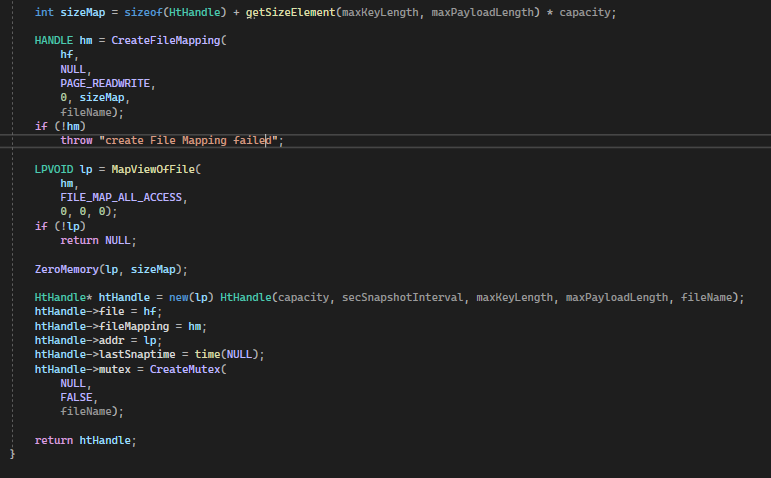
Возвращается указатель на дескриптор, потому что дескриптор в данном случае является структурой, а указатель на нее используется в качестве единого идентификатора хранилища

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.

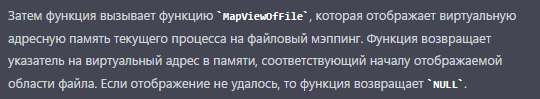
Размер данных для создания **createFileMapping:**

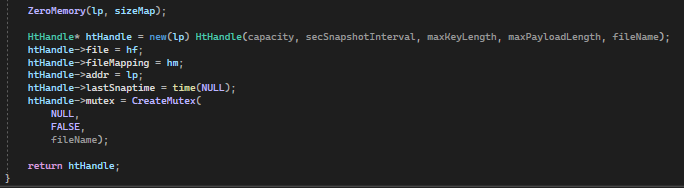






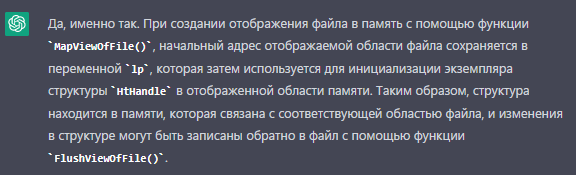




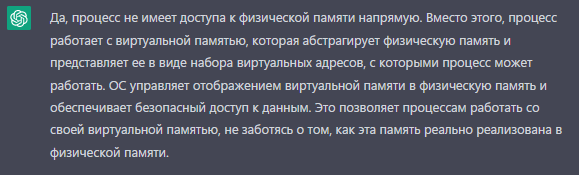


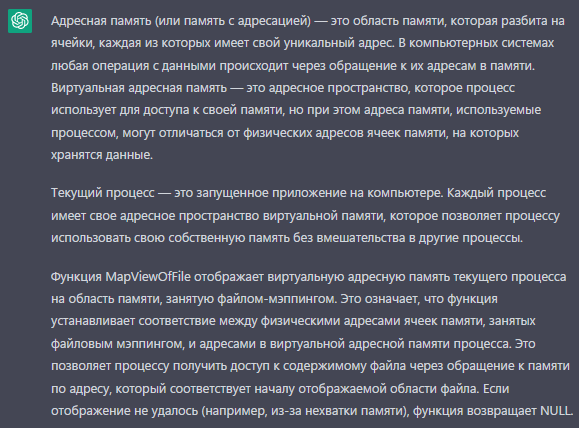
Обнуляем место в адресном пространстве на которое указывает указатель lp, создаем экземпляр структуры **new(lp)** внутри памяти адресного пространства на которое указывает lp.

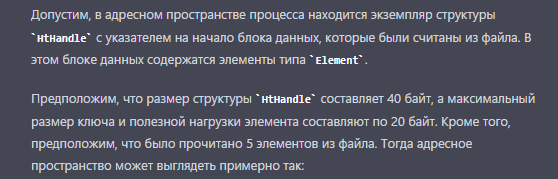
Теперь экземпляр структуры связан с файлом на диске с помощью проецирования файла в адресное пространство процесса.

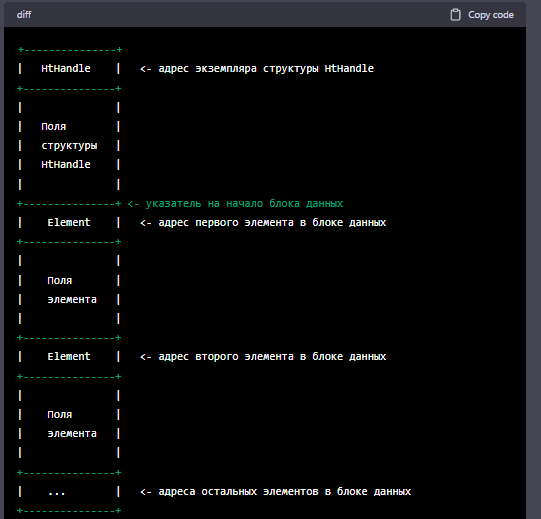


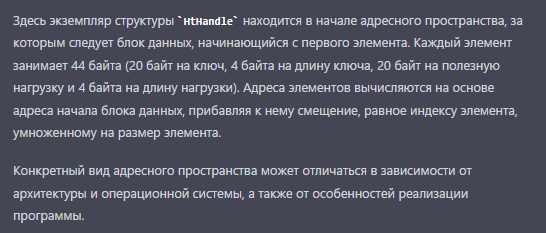
**===Адресное пространство процесса**







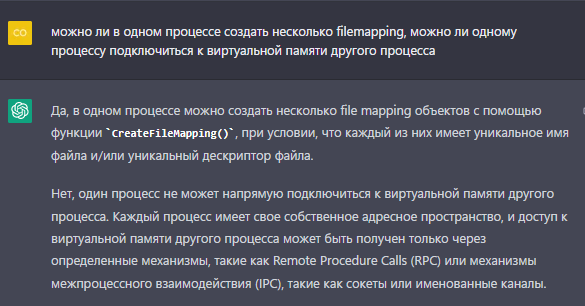




**Т.е в памяти так:**

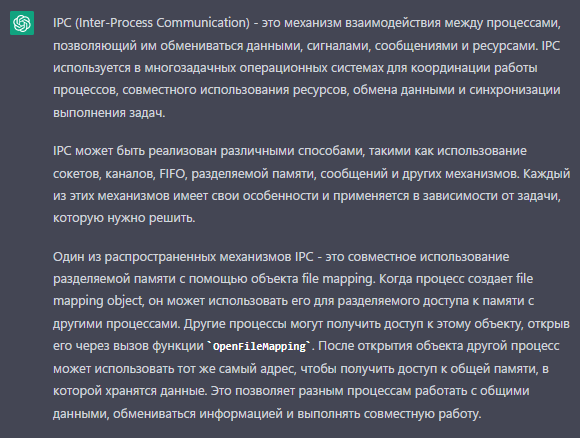
Экземпляр структуры htHandle (для доступа к полям которые содержат инфу об хеш таблице) -> элементы структуры (ключ, длина ключа(int),данные, длина данных(int)

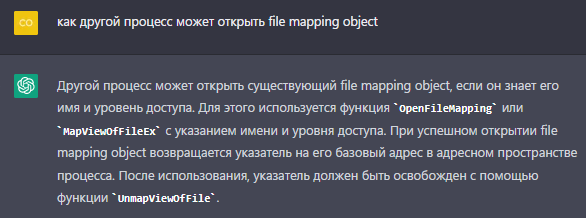
**===Можно ли в одном процессе создать несколько filemapping, можно ли одному процессу подключиться к виртуальной памяти другого процесса** (один из вопросов)



У каждого уникальный дескриптор должен быть

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



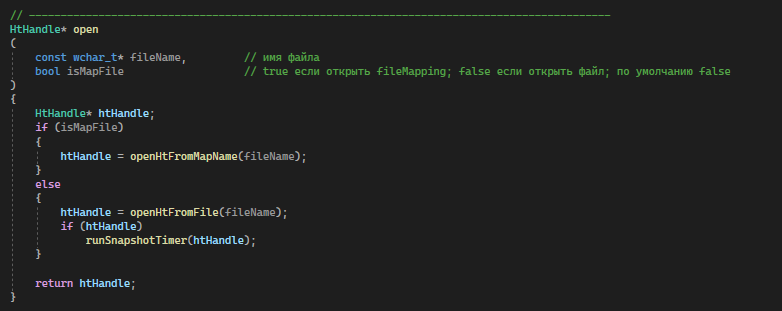


1 процесс создает FileMapping object именованный, 2 процесс подключается по имени

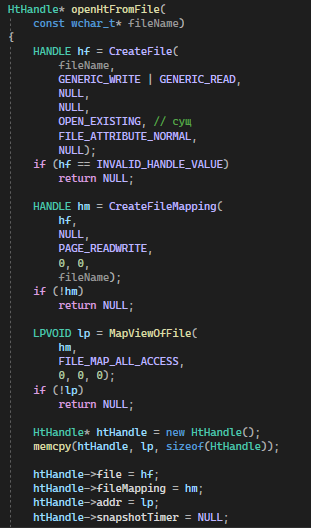
С помощью **CreateFileMapping**(указываем имя) либо с помощью **OpenFileMapping**(указываем имя)

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

**Open**



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



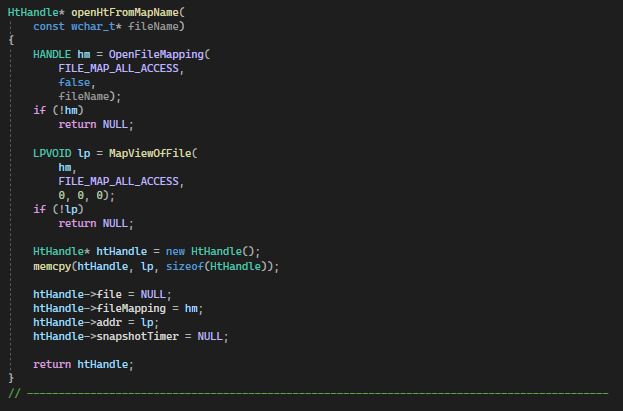
1. Open existing – открыть сущ (без создания нового)

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:**

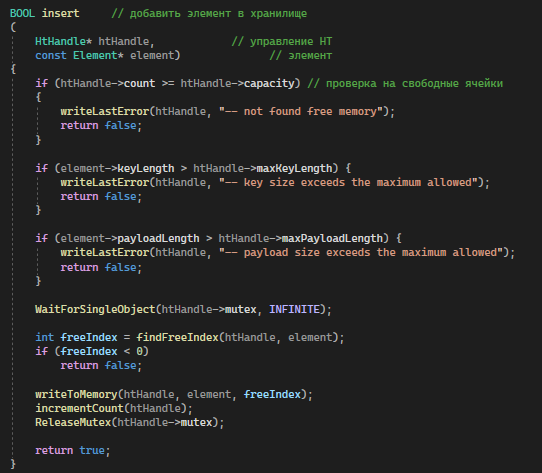


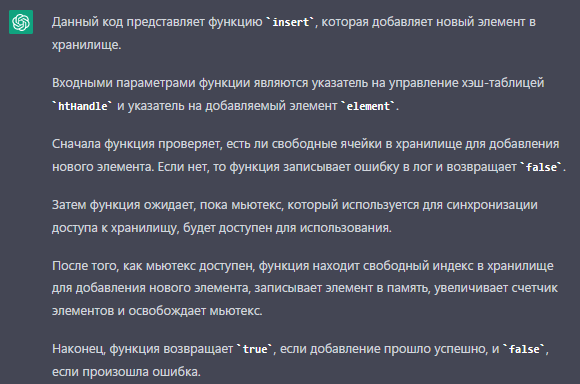
Если же у нас File mapping уже есть, то мы можем открыть его по имени и работать с теми же данными

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.

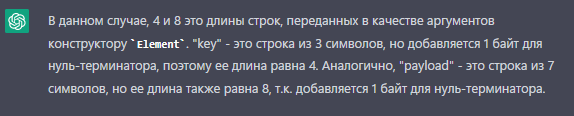
**Insert**

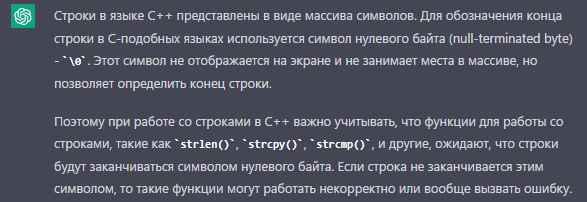






New Element() возвращает указатель





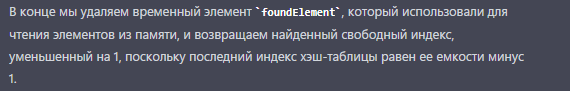


**FindFreeIndex**

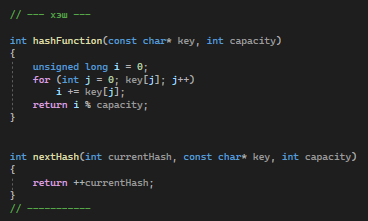


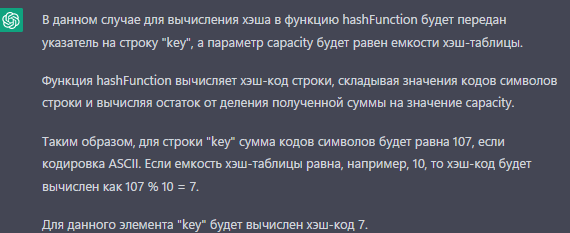
Тут происходит реализация открытой адресации, линейный перебор

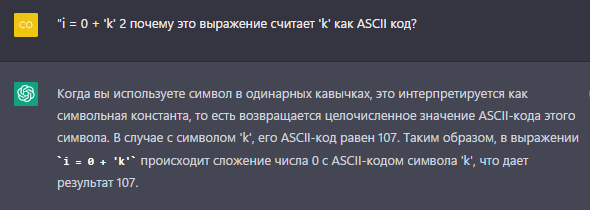




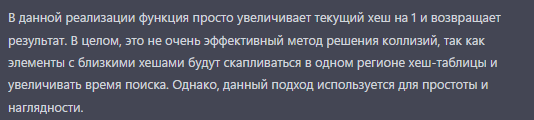
**Хэш функции**

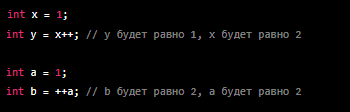






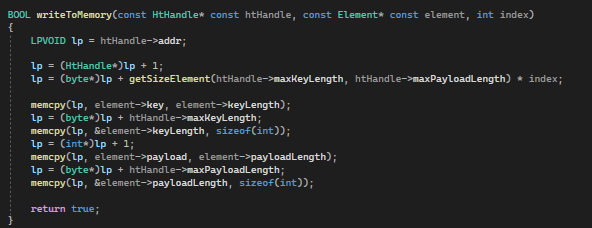




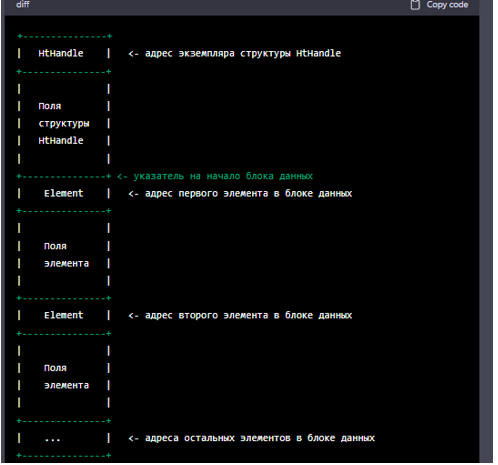


**writeToMemory**

заполнение памяти(поля элемента) по указателю

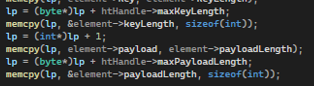


1. Получаем указатель на область памяти где хранится хэш таблица **htHandle -> addr;**
2. Увеличиваем указатель на размер структуры (HtHandle\*)lp + 1 , (тк в начале памяти хранится сама структура, потом ее элементы)



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

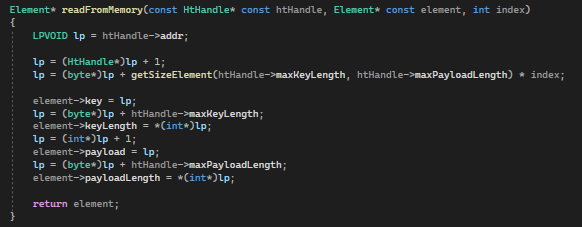
1.  заполняем этот блок памяти ключом который мы передаем в функцию
2. 

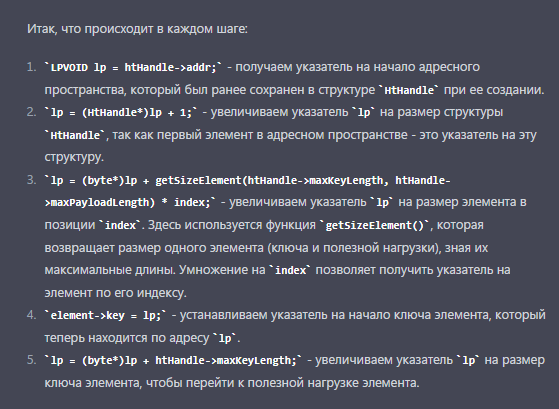
Перемещаемся к след блоку памяти добавляем **maxKeyLength** , получаем блок памяти который содержит **поле длина ключа** и записываем в этот блок данные

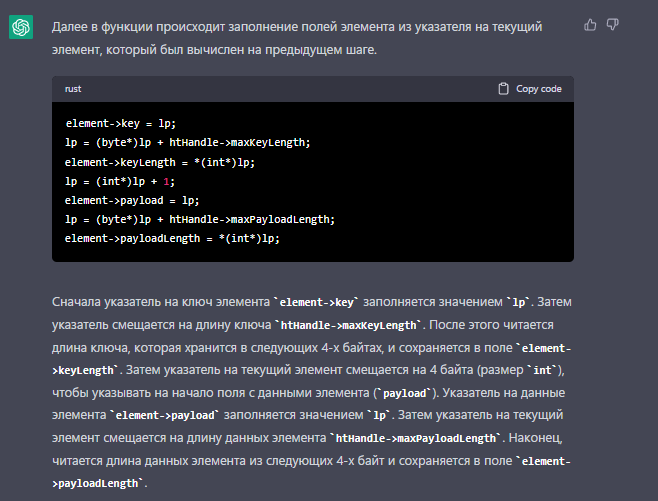
1. lp (int\*)lp + 1 – **сдвиг на 4 байта (размер инт)**
2. Получаем **поле payload данных**, записываем в память
3. Сдвигаем на размер данных и получаем **поле payloadLength длина данных**

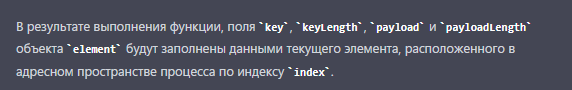
**readFromMemory**

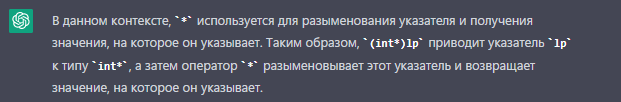
чтение памяти(полей элемента) по указателю





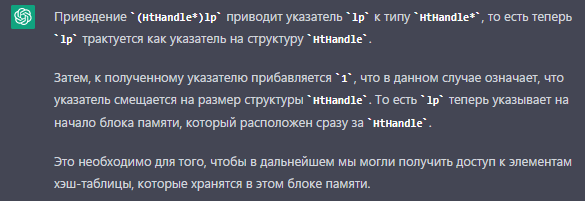




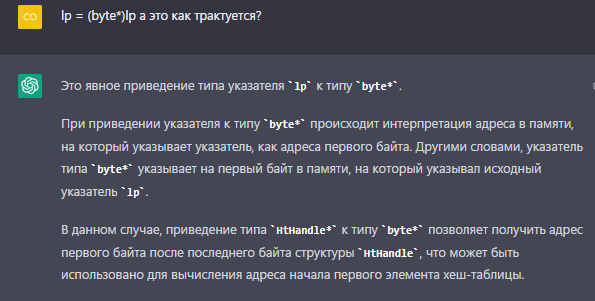


разыменование

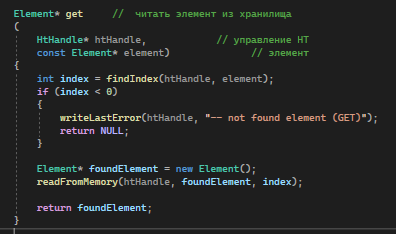


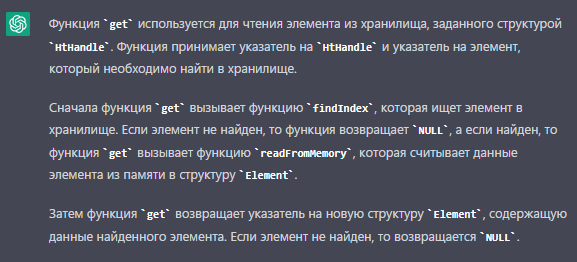




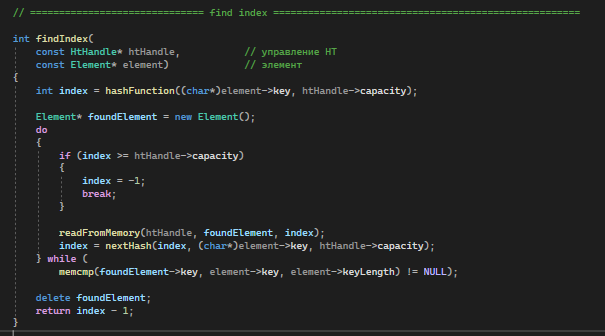


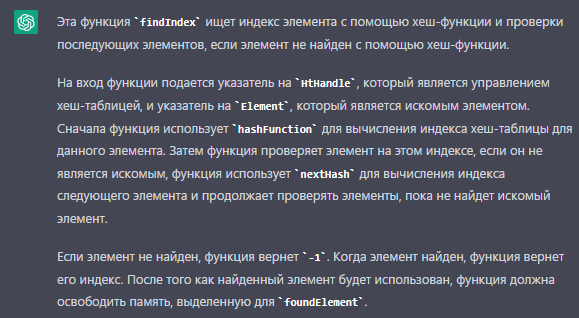
**Get**



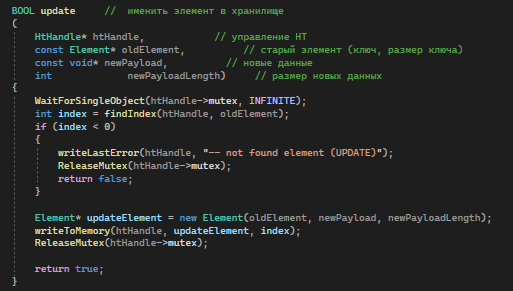


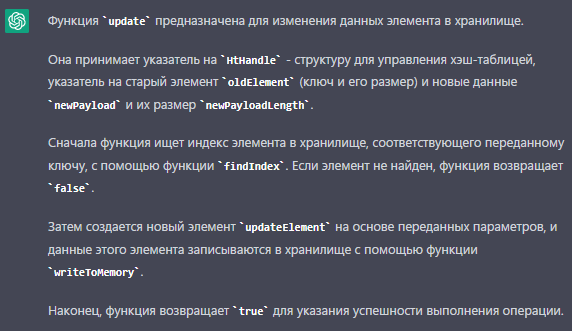
**FindIndex**



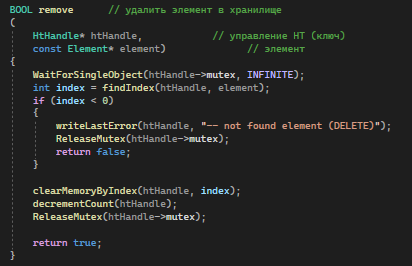


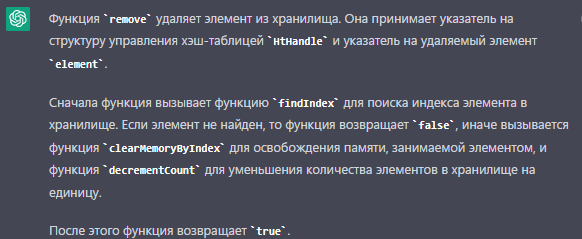
**Update**



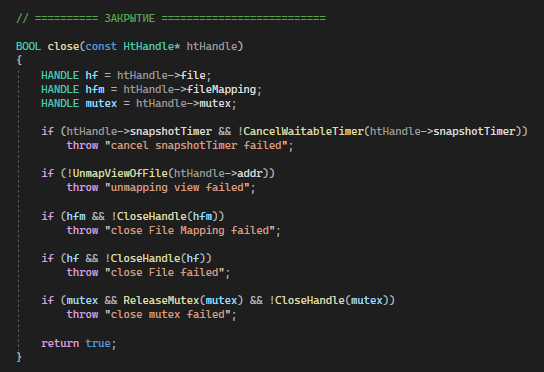


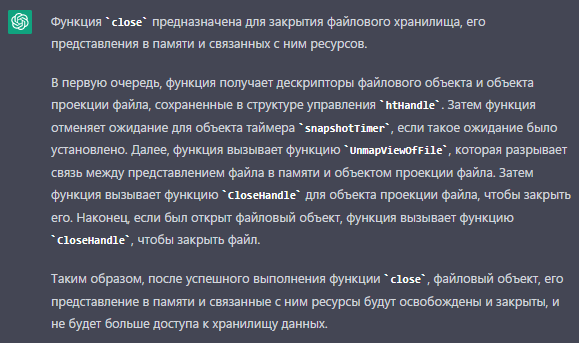
**Remove**





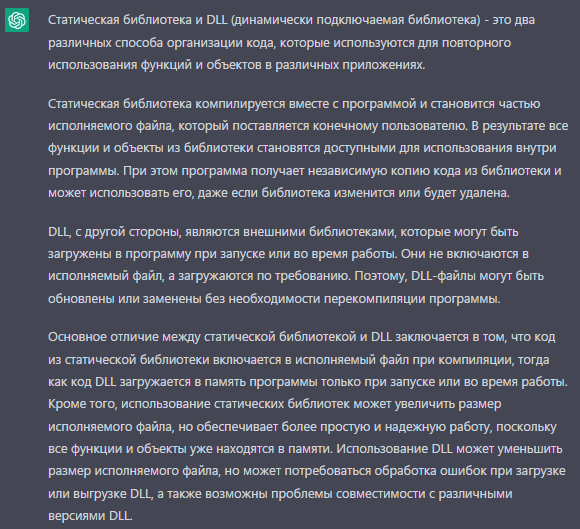
**Close**

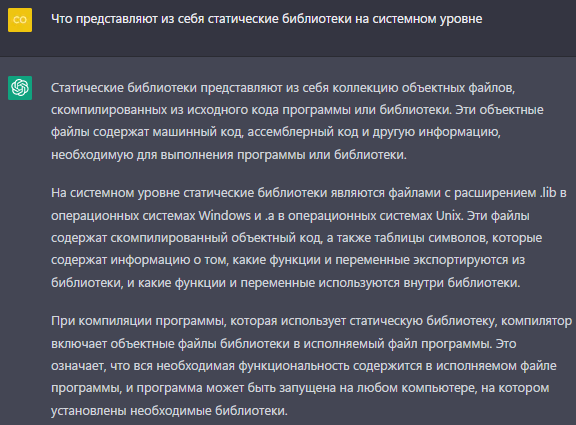


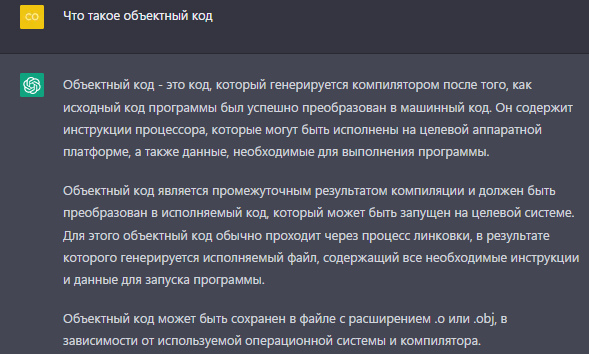


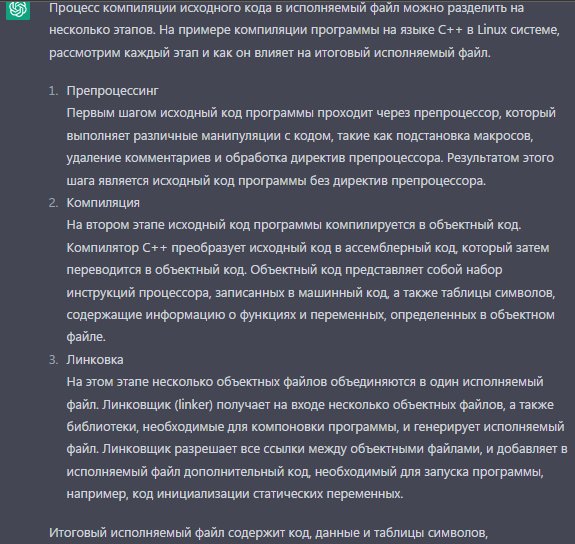
**Статическая библиотека**

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



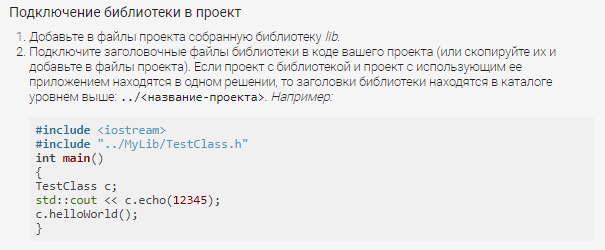






Был вопрос на каком этапе подкл статическая библиотека -> на этапе линковки (объединения в исполняемый файл)





**Задания:**

**Задание 01.Windows**

1. Разработайте приложение **OS10\_01**.
2. Приложение **OS10\_01** должно реализовывать **HT API**.
3. Приложение **OS10\_01** должно демонстрировать выполнение всех функций **HT API**.

**Задание 02.Windows**

1. Разработайте *статическую библиотеку* **OS10\_HTAPI**.
2. Разработайте приложение **OS10\_02,** функционально**-**повторяющее **OS10\_01,** но использующее библиотеку **OS10\_HTAPI**.
3. Продемонстрируйте работоспособность приложения **OS10\_02.**

**Задание 03.Windows**

1. Разработайте приложение **OS10\_03,** применяющего библиотеку **OS10\_HTAPI** и использующее одновременно 2 экземпляра HT-хранилища.
2. Продемонстрируйте работоспособность приложения **OS10\_03.**

**Задание 04.Windows**

1. Разработайте **Test Case** **HT API.**
2. Разработайте **тесты, соответствующие Test Case HT API**. (тесты 4 положительных 4 негативных)
3. Продемонстрируйте выполнение тестов.