File Mapping

ПРО МЬЮТЕКС

Значения параметров из бота

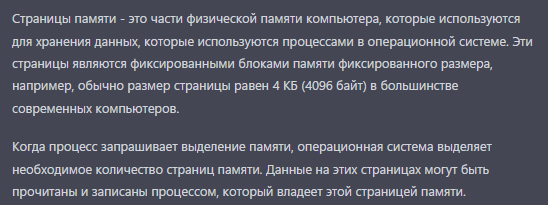
1. ***File Mapping*** – это связь содержимого файла с частью виртуального адресного пространства процесса; отображение файла в виртуальную память; позволяет работать с памятью (*адресным пространством*), а не с файлом.

Или

– это ссылка на весь файл или на некоторую часть, которая проецируется в физическую память.

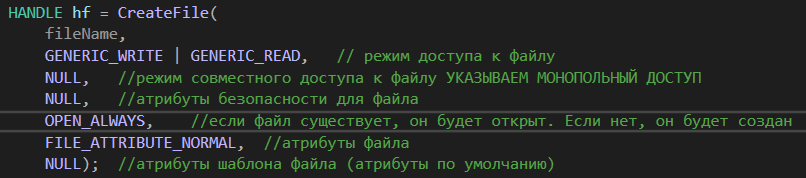
***File View*** – это часть виртуального адресного пространства, которую процесс использует для доступа к содержимому файла; отображенный файл (представлением файла). Является *указателем* на определенную часть File Mapping объекта

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



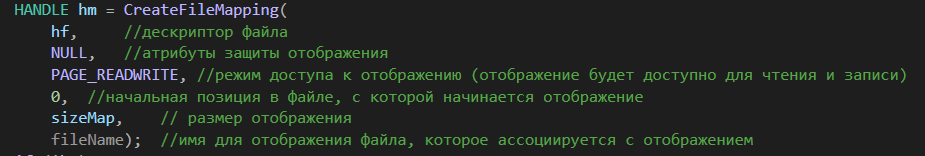
# По шагам:

1. Создать файл **CreateFile**



1. Создание отображения файла(File Mapping Object): **CreateFileMapping.** File Mapping Object находится в физической памяти (в оперативной памяти)

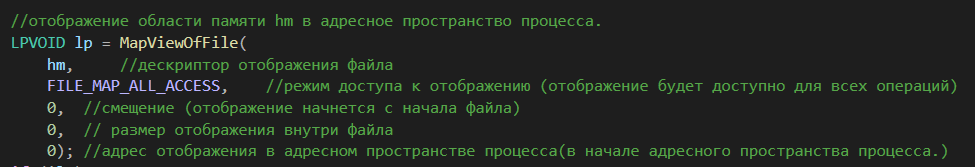
--Про функцию



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).

1. Создать представление файла **MapViewOfFile.** Отображение в виртуальное адресное пространство процесса.То есть работа с файлом будет осуществляться через указатель на память(lpvoid,int\* и тд).

**--про функцию**

****

Последний параметр – количество байт, которые нужно отобразить в виртуальное адресное пространство процесса, т.к. размер файла больше нуля то отобразиться весь файл

Размер кратен страницам памяти

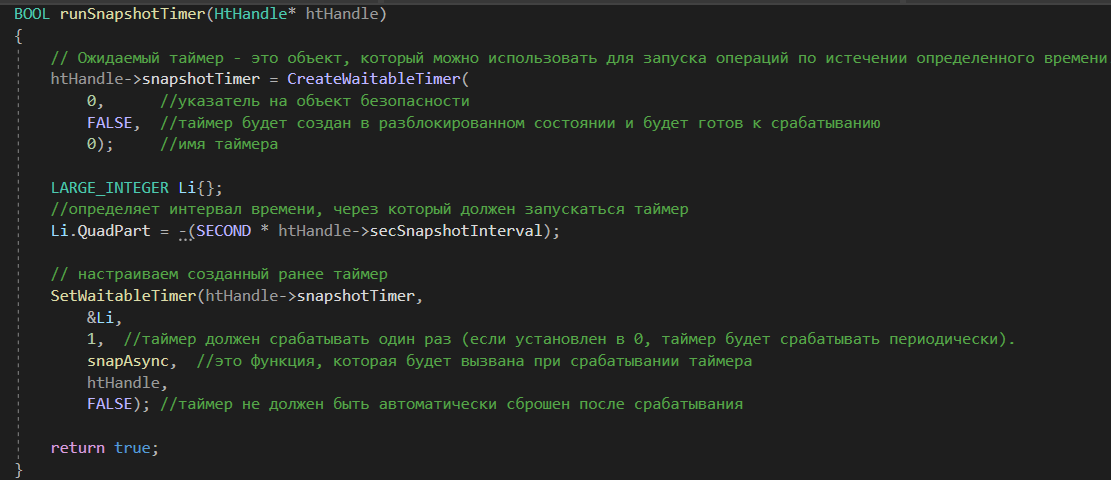
возвращает LPVOID указатель, возвращает УКАЗАТЕЛЬ НА ПАМЯТЬ

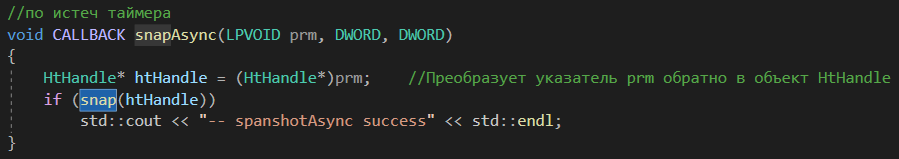
1. **Обмен данными через отображение**

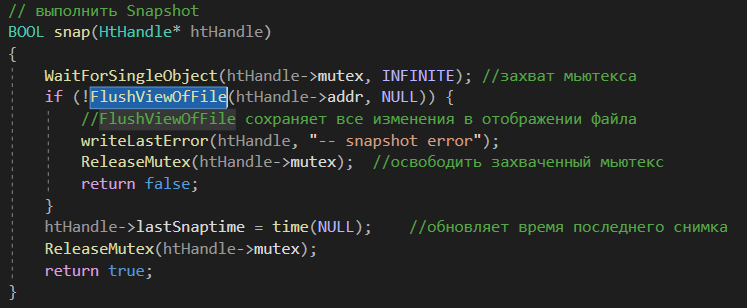
Для записи и чтения используем указатель

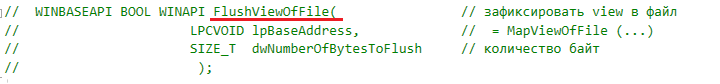
1. **Какая-то бурда с FlashViewOfFile в функции snap**

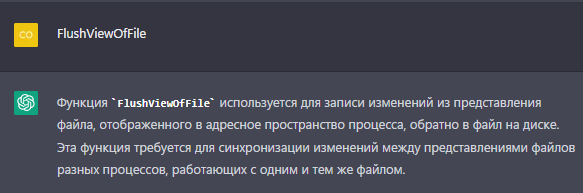
**false –** таймер в синхронном режиме







****



1. **Закрытие отображения и объекта файла: UnmapViewOfFile и CloseHandle**

очищает адресное пространство процесса

***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.

Разница физической и виртуально памяти:

Физическая память представляет собой реальные ячейки памяти на физических устройствах, таких как RAM (оперативная память) и диски. Физическая память служит для непосредственного хранения данных и исполняемого кода во время работы компьютера.

Виртуальная память позволяет каждому процессу иметь свой собственный адресный диапазон, изолированный от адресных диапазонов других процессов. Это обеспечивает изоляцию и безопасность выполнения программ

2. *Адресное пространство процесса* – это диапазон виртуальных адресов, который выделен для определенного процесса в операционной системе. Размер адресного пространства зависит от разрядности процессора и операционной системы. В 32-битных системах адресное пространство ограничено 4 ГБ, в то время как в 64-битных системах оно может быть значительно больше.

3. как происходит межпроцессное взаимодействие с помощью filemapping

*Межпроцессное взаимодействие* (IPC) с использованием File Mapping в Windows предоставляет способ для нескольких процессов разделять области памяти и обмениваться данными. Вот общий порядок действий:

1. **Создание File Mapping:**
   * Один процесс создает объект File Mapping с помощью функции **CreateFileMapping**. Этот процесс становится владельцем объекта File Mapping.
   * Другие процессы, которые хотят участвовать в межпроцессном взаимодействии, могут открыть существующий объект File Mapping с помощью функции **OpenFileMapping**. Они используют имя объекта File Mapping, чтобы найти нужный объект.

**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).**

1. **Отображение виртуальной памяти:**
   * Владелец объекта File Mapping использует **MapViewOfFile** для отображения объекта в своем виртуальном адресном пространстве. Это дает ему указатель на область памяти, которая считывает или записывает данные.
   * Другие процессы могут также использовать **MapViewOfFile**, чтобы отобразить тот же объект File Mapping в своем виртуальном адресном пространстве.
2. **Обмен данными:**
   * Процессы могут обмениваться данными, записывая и читая из отображенной области памяти.
   * Поскольку все процессы используют один и тот же объект File Mapping, любые изменения в одном процессе мгновенно отражаются в других процессах, которые отображают тот же объект.
3. **Снятие отображения и закрытие объекта:**
   * По окончании взаимодействия каждый процесс должен снять отображение с помощью **UnmapViewOfFile**.
   * Владелец объекта File Mapping может закрыть его с помощью **CloseHandle**. Это не удаляет объект, но означает, что он не может быть использован другими процессами, открывающими объект позже.
4. **Удаление объекта File Mapping:**
   * Если объект File Mapping больше не нужен, владелец может его удалить с помощью **CloseHandle**. Это приведет к удалению объекта после того, как все процессы закроют все связанные с ним отображения.

ИЛИ

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.

4. Хеш-таблица

**Хеш-таблица** — это структура данных, в которой все элементы хранятся в виде пары ключ-значение, где:

* **ключ** — уникальное число, которое используется для индексации значений;
* **значение** — данные, которые с этим ключом связаны.

5. Коллизия – это ситуация, когда хеш-функция генерирует один индекс для нескольких ключей и возникает конфликт: неизвестно, какое значение нужно сохранить в этом индексе.

6. Методы решения коллизии

* метод цепочек – если хеш-функция выделяет один индекс сразу двум элементам, то храниться они будут в одном и том же индексе, но уже с помощью двусвязного списка.
* метод открытой адресации: линейное и квадратичное зондирование.

Про метод открытой адресации и его виды

**a) Линейное зондирование**

Линейное зондирование решает проблему коллизий с помощью проверки следующей ячейки.

Проблема линейного зондирования заключается в том, что заполняется кластер соседних ячеек. Это приводит к тому, что при вставке нового элемента в хеш-таблицу необходимо проводить полный обход кластера. В результате время выполнения операций с хеш-таблицами увеличивается.

**b) Квадратичное зондирование**

Работает оно так же, как и линейное — но есть отличие. Оно заключается в том, что расстояние между соседними ячейками больше (больше одного)

при попытке добавить элемент в занятую ячейку *i* начинаем последовательно просматривать ячейки *i*+1,*i*+4,*i*+9

**c) Двойное хэширование**

Используется 2 различных хеш-кода для поиска свободной ячейки

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

Главное отличие между статической и динамической библиотекой заключается в том, когда код из библиотеки интегрируется в исполняемый файл:

1. **Статическая библиотека:**
   * **Интеграция на этапе компиляции:** Код из статической библиотеки копируется в исполняемый файл на этапе компиляции. Исполняемый файл содержит полный код, необходимый для выполнения программы.
2. **Динамическая библиотека:**
   * **Интеграция во время выполнения:** Код из динамической библиотеки загружается в память во время выполнения программы. Исполняемый файл содержит информацию о том, где и как загрузить динамическую библиотеку, но сам код библиотеки не включается в исполняемый файл.

***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***.

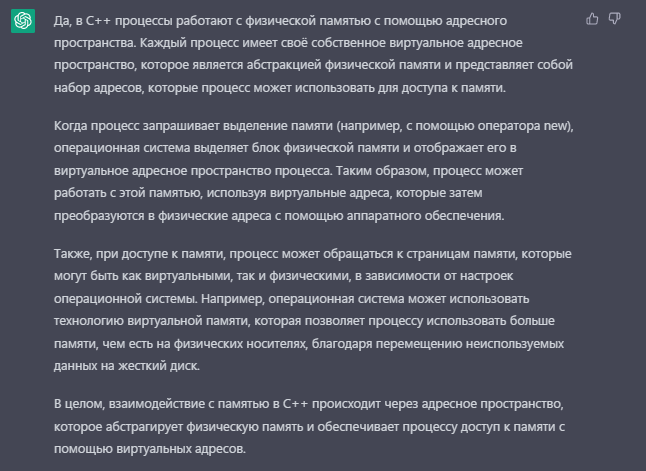
Данные образа организованы в виде хэш-таблицы с открытой адресацией.

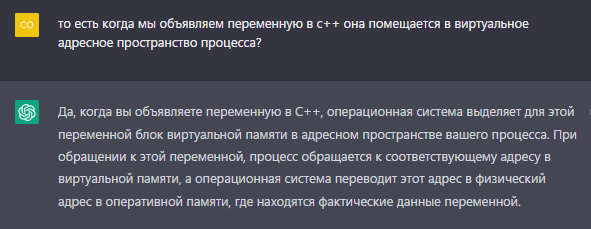
Сохранность данных осуществляется с помощью асинхронного *snapshot*- механизма (с заданным периодом асинхронное сохранение образа в файле), *snapshot* может выполняться синхронно по команде пользователя.

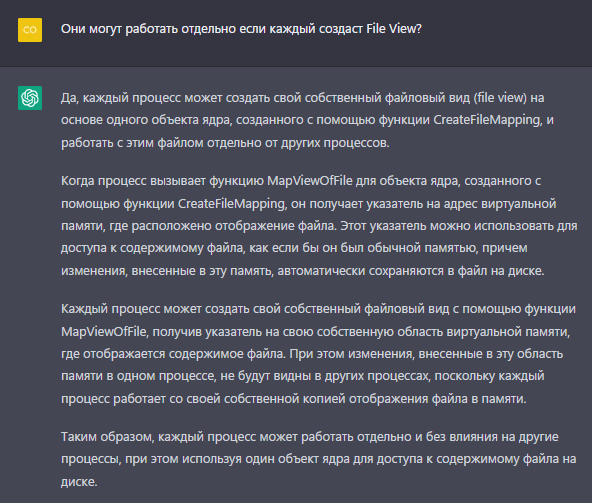
Для использования HT в приложении предназначено потокобезопасное HT API.

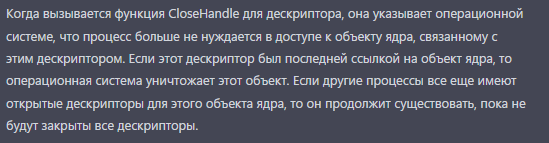
***Пусть будет***

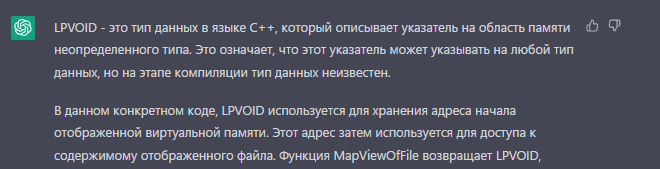
Процесс может получить доступ к памяти другого процесса ТОЛЬКО через механизмы межпроцессорного взаимодействия

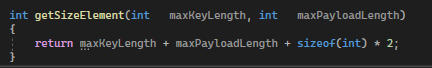




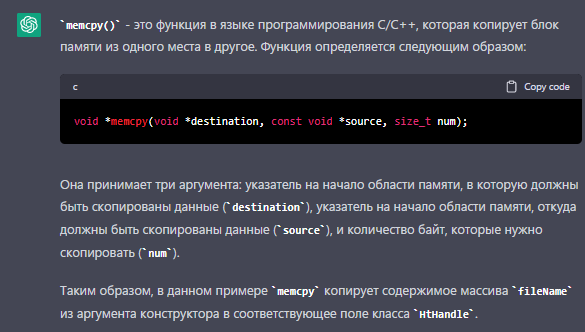


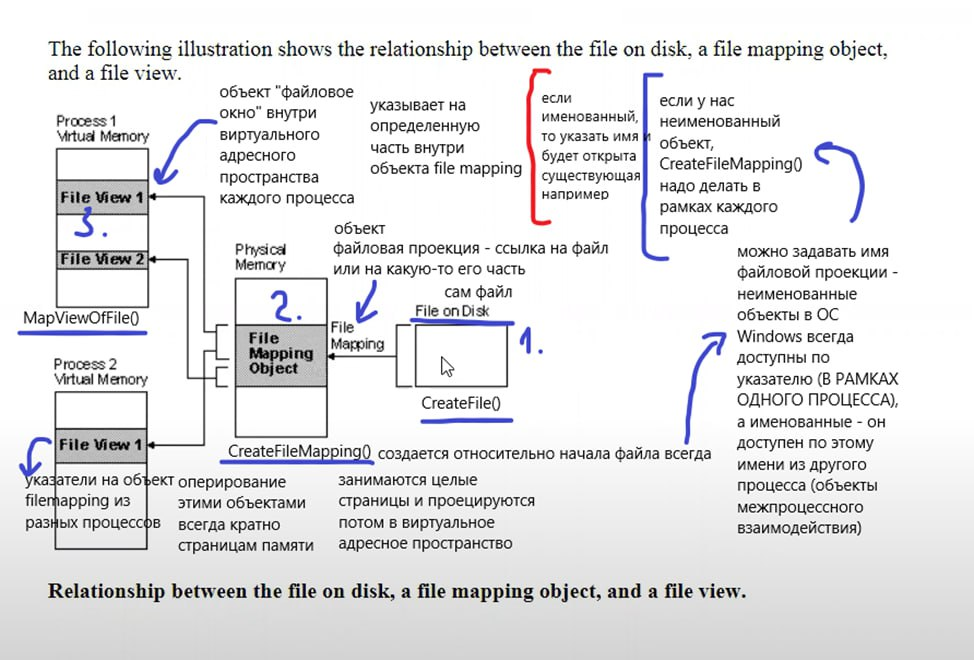






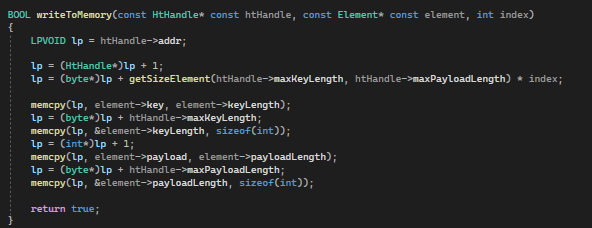
Размер 1 элемента = макс длина ключа + макс длина данных + 2 инта для хранения keyLength и payloadLength



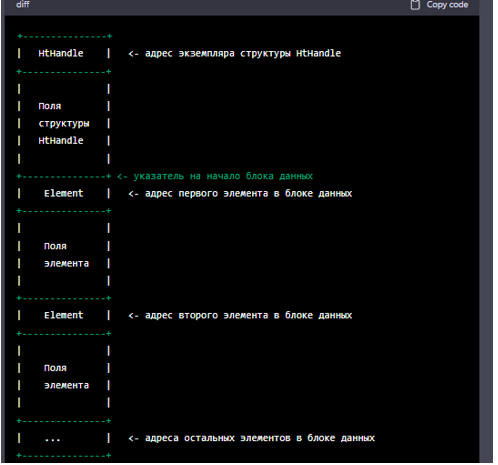


**writeToMemory**

заполнение памяти(поля элемента) по указателю

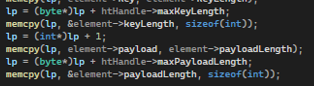


1. Получаем указатель на область памяти где хранится хэш таблица **htHandle -> addr;**
2. Увеличиваем указатель на размер структуры (HtHandle\*)lp + 1 , (тк в начале памяти хранится сама структура, потом ее элементы)



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**

***Всякие ссылки***

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

Исправление удаления  
<https://neerc.ifmo.ru/wiki/index.php?title=%D0%A0%D0%B0%D0%B7%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BE%D0%BB%D0%BB%D0%B8%D0%B7%D0%B8%D0%B9>