**1. Windows HOOK. Назначение и особенности использования.**

Хуками называются перехватчики системных событий. Они позволяют программно подменять обработчик этих событий (накладывать свои функции вместо системных).

Сама работа ХУКА направлена на получение системных событий. Поэтому необходимо обрабатывать список входящих системных событий.

Для того чтобы интерпретировать события нажатия виртуальные клавиши используется функция TranslateMessage (оно принимает событие нажатия на клавишу, например).

Она возвращает логическое значение и принимает в качестве аргумента указатель на структуру MSG (то есть системное сообщение, событие).

За перевод системного события отвечает TranslateMessage. Она принимает указатель на структуру MSG (системное событие), а возвращает числовой код или символ, который уже можно использовать в программе.

**2.       Актуальность синхронизации потоков при параллельной обработке данных**

В случае, когда два или более потоков пытаются получить одновременный доступ к какому-либо общему ресурсу (например, к данным в оперативной памяти), поведение программы может быть неверным. Значение может быть записано раньше времени или прочитана раньше времени (так как два потока получают доступ к области памяти в рандомное время)

Существует несколько способов синхронизации потоков, однако все они в той или иной мере способствуют запрещению параллельного доступа к общему ресурсу.

Есть способ синхронизации «Критическая секция».

Критическая секция -  это участок кода, который фиксируется для выполнения только одним потоком (то есть может выполняться только одним потоком одновременно). Пока критическая секция занята одним потоком, другие ждут своей очереди и не взаимодействуют с ним.

**3.       Библиотеки DLL. Назначение и использование DLL-библиотек**

По мере усложнения проекта количество используемых функций возрастает, и не всегда использование Многофайловых проектов решает эту проблему. Использование множества функций в одном проекте влечет за собой следующие проблемы:

* Не все функции используются одновременно, но они компилируются все вместе, чем занимают память. Они не нужны в данный момент, но все равно занимают место, как, например, игрушки в телефоне, в которые ты редко заходишь).
* Если одни и те же функции необходимо использовать в разных проектах, то код приходится копировать. Это неудобно и занимает больше времени, чем использование длл

Динамические библиотеки подключаются к проекту и отключаются в тот момент времени, когда это надо. Таким образом достигается экономия оперативной памяти.

Сами динамические библиотеки содержать в себе те же функции и код, который можно использовать. Туда разумно помещать функции, которые используются часто в разных проектах. Или же туда можно запихивать функции, которые используются редко, чтобы они не занимали место в оперативной памяти при работе программы.

При подключении длл библиотек нужно

1. Подключить DLL. (Создается дескриптор данной библиотеки)
2. Импортируется функция из этой библиотеки
3. Выполнить эту функцию в коде
4. Отключить библиотеку (освободить память дескриптора библеотеки)

Нужно обязательно указать, какие функции будут экспортироваться из длл (выносится в проект), а в основной программе указать, какие функции нужно импортировать из длл (вытащить оттуда). Причем, нужно, чтобы это совпадало: если импортируешь функцию в программу, нужно, чтобы в длл она была указана, как экспортируемая.

Кстати, можно подключить библиотеку WinAPI к программе на языке C#. Это ускоряет время выполнение самой программы, но средства языка C# не смогу анализировать функции библиотеки и предотвращать проблемы.

**4. Именованные каналы. Назначение и использование**

Канал (англ. Pipe) - область виртуального пространства, которое может быть использовано для совместного доступа различными процессами (своего рода виртуальный файл, в который разные программы могут записывать данные и читать их, обмениваться этими данными)

Однако канал не может храниться отдельно от какого-то процесса. Поэтому информация о нём будет очищена если завершить процесс (или программу), который его создал.

Будем называть процесс, который создаёт канал Сервером, а процессы, которые подключаются к каналу - Клиентами.

Каналы имеют несколько разновидностей:

* Симплексные - это однонаправленные. Например, Cервер только записывает данные, а Клиенты только читают их. Или клиенты только пишут, а сервер только читает.
* Дуплексные - это когда и Клиенты, и Сервер могут и читать, и писать
* Бинарные или текстовые. По аналогии с бинарными или текстовыми файлами
* С общим или разделяемым доступом к содержимому
* Именованные или анонимные

Именованные каналы помимо дескриптора имеют также Имя в виде строки. Причём это имя должно всегда оставаться одним и тем же, как у Сервера, так и у Клиента

Имя является сетевым (доступно другим компьютерам, что позволяет обменивать данными не только разным процессам, но и, по сути, разным машинам).

Простое использование именованного канала – обмен сообщениями (простой мессенджер). Сервер и клиент могут писать друг другу сообщения в виде строк, читать их и отвечать (если канал работает в дуплексном режиме).

Остальные способы использования зависят от контекста: можно передавать и получать системные данные, тот же кейлоггер сделать, обмениваться результатами вычислений и т.п.

**5. Линейный односвязный список. Особенности создания и примеры использования**

Линейный односвязный список – это более удобный (для компьютера) массив. По сути, это набор структур: в первом поле содержится какое-то значение, во втором содержится указатель на следующий элемент. В отличии от обычных массивов, односвязный список не записывается в память подряд, элементы списка могут находится в разных областях памяти. Еще одно преимущество – список можно дополнять, достаточно просто записать еще одно новое число, а предыдущему дать указатель на это новое число.

Пример: числовой односвязный список. Создаем структуру с двумя полями: первое поле – числовое, второе поле – указатель. В первое поле будем записывать число, во второе – указатель на следующий элемент.

[1, указатель на второй элемент] -> [2, указатель на третий элемент] -> и т.д.

Кстати, у односвязных списков есть и свой недостаток: чтобы получить доступ, например, к 256 элементу, нужно пройтись по всем предыдущим, так как путь к нужному элементу известен только тому, что стоит перед ним (в нашем случае 255 элементу).

**6.       Массив как тип данных. Особенности использования массивов при разработке программ на си**

Фактически массив является указателем на последовательность элементов определенного типа. Имя массива является указателем на его первый элемент. А объём памяти равен суммарному объему всех элементов с учетом их типа. Чтобы выделить память под массив, необходимо найти область в оперативной памяти, которая будет равна объему массива (то есть все элементы хранятся в памяти в одной области, в отличие от односвязного списка).

По сути, массив – это множество элементов одного типа. Доступ к каждому элементу можно получить не перебирая все предыдущие элементы (как с односвязным списком), достаточно указать его номер (например, Massive[35] – это 35 элемент массива). У массивов фиксированная длина, которая задается во время определения массива, эта длина постоянна, и менять ее нельзя.

**7.       Многопоточный режим работы компьютера. Преимущества использования многопоточности**

Многопоточность — свойство платформы (например, операционной системы) или приложения, состоящее в том, что процесс, созданный в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть каждый со своим временем, без строгих рамок.

По-настоящему многопоточный режим может достигаться только на много процессорных системах, так как только несколько процессоров могут выполнять задачи параллельно. На однопроцессорных же системах применяется псевдопаралелльный метод – потоки все равно выполняются друг за другом, но делятся на более мелкие куски, что позволяет, например, двум потокам выполнятся как бы параллельно. Это не особо сокращает общее время выполнение обоих процессов, но позволяет им выполнятся в одном темпе, что в некоторых случаях может оказаться полезным (например, если мониторить(следить) несколько показателей в реальном времени).

Преимущество многопоточного режима в одном процессе – выполнение нескольких потоков одновременно, параллельно. Несколько потоков будут выполнятся одновременно, что сократит общее время выполнения процесса.

**8.       Особенности использования строкового типа данных. Массивы строк**

Строковые типы данных необходимы в первую очередь пользователю, нежели человеку. Строка – это массив символов, последним элементом которого является закрывающий ноль.

В основном строки используются для диалога с пользователем, так как машине все равно на эти наши буквы).

Для создания массива строк используется двумерный массив символов. Левый индекс определяет число строк, а правый индекс - максимальное число символов в каждой строке. Доступ к отдельным строкам очень прост - необходимо просто определить левый индекс (то есть, номер строки).

**9.       Особенности настройки проекта и использования синтаксиса языка программирования при написании DLL-файла**

Для создания библиотеки DLL нужно добавить соответствующий проект в решение и настроить его во на запуск как динамическую библиотеку.

Проект DLL имеет некоторые особенности.

Он настраивается также, как и проект WinAPI, но помимо этого ещё необходимо указать в настройках проекта на вкладке «Общие» -> «Тип конфигурации» -> «Динамическая библиотека», а в на вкладке «C/C++» -> «Дополнительно» -> «»

Кстати, можно подключить библиотеку WinAPI к программе на языке C#. Это ускоряет время выполнение самой программы, но средства языка C# не смогу анализировать функции библиотеки и предотвращать проблемы.

Точка входа у библиотеки DLL тоже своя:

BOOL WINAPI DllMain(HINSTANCE hlnstDll, DWORD dwReason, LPVOID IpReserved)

{

BOOL bAllWentWell = TRUE;

switch (dwReason)

{

case DLL\_PROCESS\_ATTACH:

break;

case DLL\_THREAD\_ATTACH:

break;

case DLL\_THREAD\_DETACH:

break;

case DLL\_PROCESS\_DETACH:

break;

}

if (bAllWentWell)

return TRUE;

else

return FALSE;

}

Также же функции необходимо пометить с помощью специального оператора:

* \_\_declspec(dllimport) - для Импортируемых функций
* \_\_declspec(dllexport) - для Экспортируемых функций

И это всё описывается в прототипе функции. Без прототипа импортировать или экспортировать ее низя.

Для импорта функции необходимо сначала создать указатель на функцию с сигнатурой, Которую мы хотим вызвать. Также мы должны указать соглашение о вызовах. (\_cdecl).

За подключение библиотеки отвечает функция LoadLibrary (ей передается путь к длл), за отключение отвечает FreeLibrary (принимает дескриптор длл).

За инициализацию функции из DLL отвечает функция GetProcAddress. Она принимает два параметра: дескриптор DLL и строку, содержащую имя импортируемой функции. Возвращает указатель на эту функцию.

**10.   Особенности разработки программ в WinAPI**

Операционная система Windows может предоставлять интерфейс для выполнения определённых системных задач сторонними программами. Такой интерфейс называется WinAPI.

WinAPI Представляет из себя набор функций, структур и различных параметров.

Функция запуска Windows приложений называется WinMain(аргументы: дескриптор запущенного приложения, дескриптор win16(неактуален), указатель на командную строку, значение для вывода командной строки).

Перед запуском приложений Windows необходимо особым образом настроить проект: в качестве подсистемы проекта выбрать Windows и выбрать компилятор Си. Также при работе с API системы Windows необходимо подключать заголовочный файл Windows.h.

В данном случае создается системный процесс, который не имеет интерфейса. Для ввода информации обычно используются файлы или другие способы (буфер обмена, именованный канал и т.п.). Вывод информации также осуществляется в файл или другой источник данных. Также вывод может осуществляться в диалоговое окно вывода, за это отвечает функция MessageBox(аргументы: дескриптор окна(мы ставим NULL), текст сообщения, заголовок окна, параметры(кнопки, иконки и т.п.)).

**11.   Особенности разработки программ для работы с системными событиями использования мыши и клавиатуры.**

Для работы с системными событиями необходимо использовать ХУКИ. Они перехватывают системные сообщения. Так можно перехватывать нажатия на клавиши или использование мыши.

Для клавы: HHOOK hHookKeyboard = SetWindowsHookExW(WH\_KEYBOARD\_LL, LogKey, NULL, NULL);

WH\_KEYBOARD\_LL – это обработка нажатий на клавиши клавы, LogKey – функция, которая будет вызываться при нажатии на клавишу. Остальное – значения по умолчанию, поэтому NULL.

Для того чтобы интерпретировать события нажатия виртуальные клавиши используется функция TranslateMessage (оно принимает событие нажатия на клавишу, например).

Она возвращает логическое значение и принимает в качестве аргумента указатель на структуру MSG (то есть системное сообщение, событие).

Для отправки сообщения в обработчик ХУКА используется функция DispatchMessage.

Она возвращает числовой код, а в качестве аргумента также принимает указатель на структуру MSG.

Также для правильного интерпретации нажатые клавиши важно понять регистр.

Символ может быть напечатан в Верхнем регистре в двух раздельных случаях:

• Если зажата клавиша Shift

• Если нажата клавиша CapsLock

Для проверки этих ситуаций используются функция GetKeyState (она позволяет получить регистр, который потом можно преобразовать в логическое значение кнопка заглавная или Незаглавная, например).

Для перевода английской клавиша в русскую можно воспользоваться самостоятельно написанной функцией через switch (если нажата клавиша ‘q’, русской будет ‘й’ ну и так со всеми).

Полученные данные можно записать в файл, чтобы узнать, какие клавиши были нажаты пользователем. По сути, это простой кейлоггер.

Все это можно делать как с клавишами клавиатуры, так и с кнопками мыши.

Для мыши потребуется сделать свой ХУК:

HHOOK hHookMouse = SetWindowsHookExW(WH\_MOUSE\_LL, LogMouse, NULL, NULL);

Он будет перехватывать нажатие на кнопку мыши, или поворот колесика.

**12.   Подходы к обработке исключений в программах WinAPI, написанных на си**

Достаточно распространённый и простой обработать ошибки – проверить их в условном блоке. Например, многие функции возвращают логические значения, которые можно легко проверить в условии. Другие функции в случае правильного выполнения возвращают 0, иначе другие значения, что также можно записать в условие оператора if.

Такая же ситуация обстоит и с указателями. При обращении к ним, возвращается правильное значение, если все правильно, или 0, если что-то пошло не так. Собственно, достаточно просто проверить, если при обращении к указателю возвращается ноль, значит есть ошибка при чтении памяти.

**13.   Понятие “системное программирование”. предназначение и специфика системного программирования**

Системное программирование – это создание набора инструкций на языке, который будет понятен как программисту, так и системе, для которой предназначается инструкция. Системой является ОС компьютера, а программист в данном случае создает программы и утилиты, направленные на работу в ОС.

Чаще всего под системными программами понимаются те, что взаимодействуют напрямую с процессором. Эти программы направлены не на пользователя (как прикладные), а на аппаратную составляющую. Системное программирование предназначено для создания программ, которые будут взаимодействовать напрямую с машиной, выполнять различные задачи на низком уровне, облегчая (так скажем) жизнь компьютеру. Системными программами являются и драйвера, и утилиты для работы с аппаратами (например, утилита для дефрагментации диска), более комплексные программы. Все это объединяет то, что они не взаимодействуют с пользователем, максимум – интерфейс для задачи начальных параметров, все остальное будет выполнятся в ОС и не потребует вмешательства пользователя. Исходя из всего это, делаем вывод, что программист должен быть хорошо осведомлен в сфере аппаратного обеспечения, и непосредственно языка системного программирования (наш – это Си).

**14.   Предназначение директив препроцессора и заголовочных файлов**

Препроцессор - это специальная программа, которая осуществляет алгоритмические действия перед компиляцией основного кода. Команды для препроцессора называются директивами. Директивы начинаются с символа «#», в конце не ставится «;».

Список основных директив:

* #include. Вставляет содержимое из текстового файла (заголовочного) в то место, где прописана эта команда. Короче, позволяет использовать какие-то функции из заголовочных файлов в проге.
* #define. Имеет три основных применения:
  + Инициализация параметров (Задание флагов). Используется преимущественно для условной компиляции.
  + Задание констант. Используется в качестве альтернативы глобальным переменным (постоянные значения, которые ну никак не изменишь)
  + Задание макроопределений (макросов) (заранее заданные команды или условия, маленькие функции)
* #undef. Отменяют задание параметра (противоположность define)
* #ifdef - Условие компиляции, если определённый параметр задан (если есть какой-то define)
* #ifndef - Условие компиляции, если определённый параметр не задан (если какого-то define нет)
* #if - Инициализация условной компиляции. Далее необходимо ввести условие, используя другие Директивы препроцессора.
* #elif - Директивы для создания вложенных условий (внутри основного #if)
* #else - Ветка при ложности всех условий. Её нельзя ставить выше, чем #elif
* #error - Внесение искусственной ошибки для компиляции (для теста, например)

**15.   Принципы взаимодействия операционной системы с прикладными программами (в том числе раскрыть понятия “процесс”, “поток”, “дескриптор”)**

Windows осуществляет обращение к прикладным программам. Этот процесс происходит примерно следующим образом:

1. Программа находится в ожидании
2. ОС отправляет сообщение программе на запуск
3. Программа запускается
4. Программа выполняет действия

Подробнее.

Программа находится в состоянии ожидания до тех пор, пока ОС не пошлет ей сообщение.

Сообщение передается прикладной программе посредством специальной функции, которая вызывается самой ОС.

После того как сообщение будет принято, предполагается, что прикладная программа должна выполнить соответствующее действие. Запускается процесс программы – некая абстрактность, задача, выполняемая процессором, которой необходимы ресурсы для своей работы. Процесс делится на потоки – условная единица, которая выполняется в процессоре. Каждый поток выполняется поочередно, по завершению выполнения всех потоков, завершается и выполнение процесса, а соответственно и выполнении программы. Каждый процесс на протяжении всего жизненного цикла имеет дескриптор – такой описатель, который содержит номер процесса, область памяти, где размещен сегмент кода, приоритет процесса, данные о его состоянии.

**16.   Принципы функционирования системного программного обеспечения**

Системное программное обеспечение, в отличие от прикладного, функционирует без непосредственного вмешательства пользователя, на уровне операционной системы. Такое ПО взаимодействует напрямую с ОС или еще ниже, с процессором и аппаратной составляющей ПК.

К системному ПО относят утилиты, драйвера, программы, встроенные в аппаратное оборудование. Они, чаще всего, не взаимодействуют с пользователем и содержат инструкции, направленные к самой машине или ОС.

**17.   Реестр Windows. Использование системного реестра программами**

База данных, которая хранит в себе параметры и настройки большинства программ, включая системные. Реестр представляет собой иерархическую структуру. Включает пять основных ключей, которые, в свою очередь, содержат в себе другие ключи и т.д. Все ключи имеют некоторые параметры разных типов (числовые, строковые или логические), которые и являются параметры программ.

Главная цель реестра – хранить параметры всех программ в одном месте, чтобы при надобности, программы могли получать к ним доступ, читать или изменять.

**18.   Системный буфер обмена. Особенности взаимодействия программного обеспечения с системным буфером обмена.**

Системный буфер обмена — это область памяти, предоставляемая операционной системой для временного хранения тех или иных данных. Доступ к нему может получить любая программа. Каждая программа с помощью простых функций и интерфейсов может получить доступ к буферу обмена, просмотреть данные, которые в нем хранятся, захапать их себе, если вдруг они потребуются, или записать свои данные, которые смогут прочитать другие программы.

Буфер обмена – это простой способ приложениям обмениваться данными, не записывая их в файлы.

**19.   Создание процессов. Функция запуска. Аргументы командной строки**

**20.   Структура системного реестра. Типы данных в системном реестре.**

Всего в реестре Windows есть пять основных ветвей:

* HKEY\_CLASSES\_ROOT, (раздел для типов файлов, которые нужны винде)
* HKEY\_CURRENT\_USER, (инфа о пользователе, который зашел в данный момент)
* HKEY\_LOCAL\_MACHINE, (инфа об оборудовании ПК)
* HKEY\_USERS, (инфа обо всех пользователях, которые зареганы на этом пк)
* HKEY\_CURRENT\_CONFIG. (настройка оборудования для текущего сеанса использования пк)

Каждая ветвь содержит множество различных ключей, отвечающих за группы программ и сами программы. Ключи, в свою очередь, содержат параметры программ разных типов, таких как:

* REG\_BINARY - Двоичные данные
* REG\_DWORD - Число
* REG\_QWORD - 64-разрядное числовое значение
* REG\_EXPAND\_SZ - Текст и переменные
* REG\_FULL\_RESOURCE\_DESCRIPTOR - Идентификатор ресурса устройства
* REG\_LINK - Путь к файлу
* REG\_MULTI\_SZ - Массив строк
* REG\_NONE - Зашифрованные данные
* REG\_RESOURCE\_LIST - Список ресурсов устройств
* REG\_RESOURCE\_REQUIREMENTS\_LIST - Идентификатор ресурса устройства
* REG\_SZ - Текст

**21.   Структуры данных. Актуальность использования структур. Передача структуры как параметра.**

Большинство данных в компьютере можно представить в виде списка из определенных структур.

Причём хранение осуществляется именно в виде списков, а не в виде массивов, поскольку массивы имеют ряд недостатков:

* Большее количество последовательно идущих секторов памяти
* Сложность редактирования или удаления элементов в середине массива

Для того чтобы избежать эти недостатки и актуальнее использовать динамические структуры данных.

Чтобы передать структуру можно либо указать ее глобально, либо передавать указатель на ее первый элемент (он же указатель на указатель структуры).

**22.   Типы данных в ОС Windows**

* Тип BYTE обозначает 8-разрядное беззнаковое символьное значение.
* Тип WORD — 16-разрядное беззнаковое короткое целое. (положительное число)
* Тип DWORD — беззнаковое длинное целое. (большое положительное число)
* Тип UINT — беззнаковое 32-разрядное целое. (тоже большое положительное число)
* Тип LONG эквивалентен типу long. (большое число, может быть отрицательным)
* Тип BOOL обозначает целое и используется, когда значение может быть либо истинным, либо ложным.
* Тип LPSTR определяет указатель на строку.
* Тип LPCSTR определяет константный (const) указатель на строку.
* Тип HANDLE обозначает 32-разрядное целое, используемое в качестве дескриптора (описателя в котором хранится инфа о процессе, например его адрес в памяти, состояние, приоритет).

**23.   Указатель как тип данных. Особенности работы с указателями**

Указатель – переменная, содержащая адрес того или иного значения в памяти. Он используется для того, чтобы получать доступ не просто к переменной, которая хранит значение, а к самому значению прямиком в памяти. Это будет полезно, если одно значение нужно менять в разных функциях программы. Поскольку указатель – это тоже переменная, на нее также можно сделать указатель, таким образом получится указатель на указатель (а можно еще парочку добавить и будет ПИЗДОС).

Массив также является указателем на последовательность элементов. Двумерный массив – это указатель на последовательность одномерных массивов. То есть, указатель двумерного массива – это номера строк, каждая из которых тоже является последовательностью значений.

**24.   Указатель на функцию. Назначение и примеры использования**

Поскольку функция тоже занимает место в памяти, на нее также можно сделать указатель. В таком случае, при обращении к указателю на функцию будет вызываться сама эта функция. Выглядеть это будет так:

int razn(int a, int b)

{

return a - b;

}

int main(void)

{

int (\*f)(int, int);//указатель на функцию razn (она выше) с двумя аргументами типа int и возвращающую int

f = razn;//присваиваем указателю адрес функции выше

printf("%d", f(2, 3));//выводим на экран получившуюся разность (мы вызвали ее, используя указатель)

return 0;

}

Это может пригодится, когда необходимо напрямую обращаться к памяти, где хранится функция.

**25.   Файл как тип данных. Использование файлов в работе программного обеспечения**

Под файловым типом данных (файлом) понимается упорядоченная совокупность любого количества компонент одного типа. Понятие файла носит широкий характер. В качестве файла может выступать обычный файл, который хранится на жестком диске, либо коммуникационный порт компьютера, устройство печати, клавиатура и т.д.

Чаще всего, файлы используются для хранения или обмена данными между разными программами. Если на файл не наложены ограничения, каждая программа может открыть его, просмотреть данные, записать свои, или удалить их.

**26.   Функции для работы с файлами WinAPI. Синхронный и асинхронный доступ к файлу.**

За создание дескриптора файла отвечает функция Createfile

HANDLE CreateFile(указатель на имя файла (устройства), параметры доступа (можно ли читать и/или писать), разделяемый доступ (можно ли вместе читать и/или писать), безопасность (NULL по умолчанию), описание открытия (открыть готовый, создать новый и т.п.), атрибуты файла (FILE\_ATTRIBUTE\_NORMAL по умолчанию), файл шаблона (NULL если он не нужен));

За запись отвечает функция WriteFile

BOOL WriteFile(указатель на файл, указатель на буфер (откуда записываем данные в файл), объем записываемых данных (ограничение по записи), фактический размер записанных данных (может быть меньше ограничения), флаг режима доступа к файлу: асинхронный(FILE\_FLAG\_OVERLAPPED) или синхронный(NULL)).

За чтение из файла отвечает функция ReadFile. Аргументы у неё такие же как его функции WriteFile, но при чтении есть некоторые особенности:

* Надо указывать заведомо большой буфер для чтения (так как мы не знаем сколько информации находится в файле)
* После прочтения строку нужно закрыть (дописать к ней символ \0)

DWORD d = 0; //сколько фактически байт было прочитано из файла

DWORD sizeBuffer = 521;//объем буфера для чтения (ограничение)

LPSTR str = malloc(sizeBuffer+1); //строка для считывания из файла

ReadFile(hFile, str, sizeBuffer, &d, NULL); //чтение

str[d] = '\0'; //закрываем строку, добавляя в ее конец /0

При асинхронном доступе к файлу процесс продолжит выполнение, даже если чтение или запись в файл еще не закончилась. Синхронный – наоборот, соответственно, процесс продолжит выполнение после того, как взаимодействие с файлом закончится.