Экзаменационные вопросы ИСиТ-3, лето 2024

Системное программирование

1. Системное программирование: 2 определения, назначение, применение, особенности разработки приложений.
   * *в каких случаях привлекаются системные программисты?*

* *какие приложения разрабатывают системные программисты?*

1. Системный вызов: определение, назначение, применение, реализация в операционных системах, механизм исполнения.
   * *можно обойтись без системных вызовов?*

* *зачем нужны системные вызовы?*
* *что нужно сделать, что создать собственный системный вызов?*

1. Фреймворк операционной системы: определение, назначение, применение, состав. Стандарт POSIX. Фреймворки систем программирования.
   * *где дано определение понятие «фреймворк»?*

* *какая организация поддерживает стандарт POSIX?*
* *что написано в POSIX?*
* *перечислите известные фреймворки.*

1. Применение процессов в ОС Windows, API для работы с процессами.
   * *в каких случаях вы будете создавать дочерний процесс?*

* *как создать в операционной системе процесс автоматически?*
* *особенности создания процессов в linux (fork);*
* *особенности исполнения процесса в docker-окружении.*

1. Методы межпроцессного взаимодействия в ОС Windows: обмен данными, синхронизация.
   * *в каких случаях вы будете использовать IPC?*

* *можно ли организовать IPC c помощью DLL?*
* *как организовать обмен данными через память, но без файла?*

1. Применение потоков в ОС Windows, API для работы с потоками, API для синхронизации потоков.
   * *чем отличается асинхронность от многопоточности?*

* *поясните понятие «потокобезопасный объект»*
* *зачем придумали потоки?*
* *особенности реализации потоков в unix/linux;*

1. Применение механизмов синхронизации в ОС Windows, API для синхронизации.

* *основа любой синхронизации;*
* *как написать собственную синхронизацию потоков в рамках одного процесса (типа критической секции)?*
* *как реализовать синхронизацию процессов в сети.*

1. Файловая система: логическая и физическая организация данных, определение файловой системы, отличие файловых систем, оглавление файловой системы, файлы, каталоги, основные функции файловой системы, буферы ввода/вывода, кеширование ввода/вывода, основные функции API файловой системы, маркер файла, текущая позиция файла, блокировка файлов, наблюдение за изменением в каталоге, особенности устройства файловой системы в Linux.

* *что значит «монтировать» файловую систему?*
* *для чего используется файловая система proc в linux*?
* *чем отличаются файловые системы?*

1. Работа с оперативной памятью в ОС Windows: API для работы с виртуальной памятью, API для работы с Heap.

* *что такое «утечка памяти»?*
* *зачем придумали Heap?*

1. Механизм отображение файлов в памяти: последовательность системных вызовов Windows для создания образа файла в оперативной памяти, использование образа файла, как средства межпроцессного взаимодействия.

- *как можно сделать общую память для процессов без файла?*

1. Динамически вызываемые библиотеки: структура DLL-библиотеки, экспорт функций, загрузка динамической библиотеки, динамический вызов функций динамической библиотеки, создание и применение библиотеки импорта.

* *что такое «позднее связывание»?*
* *как организовать общую память для двух DLL?*
* *зачем нужны библиотеки импорта?*

*Как устроена? Что такое dll main. В каких случаях вызывается dll main (4 случая). Что такое позднее связывание и раннее. Как организовать общую память для 2 dll (в одной dll когда запускаете есть ли map view) (ну или через статическую память). Зачем нужна библиотека импорта (чтобы упростить работу, чтобы не писать loadlibrary и getprocaddr (внутри нее выполняется)). Внутри нее это выполняется. Библ импорта это делает внутри себя*

*Ответ:*

DLL (Dynamic Link Library) - это динамически подключаемая библиотека, которая содержит код и данные, которые могут использоваться несколькими приложениями одновременно. Она загружается в память при запуске приложения, и вызовы к функциям из библиотеки выполняются через указатели на функции.

**Структура DLL-библиотеки Структура DLL-библиотеки:**

- необязательную часть кода, которая отвечает за инициализацию и очистку библиотеки;

- набор подпрограмм библиотеки;

- явное указание, какие подпрограммы должны экспортироваться из библиотеки.

**Загрузка динамической библиотеки:**

* + **Явная** (через LoadLibrary, а в таблицу экспорта будут заносится только те функции, переменные и т.д., которые мы в программе загрузим явным образом)
  + **Неявная** (через свойства проекта, в таблицу экспорта будут занесены ВСЕ функции из DLL)

*Определение смелова:* DLL– программный модуль, который может быть загружен динамически и содержать функции и данные. Механизм проецирования – один и тот же экземпляр DLLиспользуется несколькими процессами (код – общий, данные по отдельности).

*ЛК:*

*DLL библиотека содержит в себе функции, обеспечивает позднее связывание (связывания на этапе выполнения). Параллельно с dll создается статическая библиотека (он имеет ввиду библиотека импорта). Статическая библиотека обеспечивает раннее связывание, считается с помощью линтера*

*В ОС фрэймворк представлен в виде динамических библиотек (фреймворк ос представлен библиотеками и функциями, которые предоставляются программисту). И у каждой библиотеки есть своя библиотека импорта.*

*В любое приложение которое работает на ОС, в него прикомпоновываются библиотеки с пом которых мы будем осущ обращения к ядру ОС библиотеки фрэймворка ОС.*

*Особенности dll:*

1. *загружается 1 раз (LoadLibrary загрузка библиотеки), все остальные случаи исп мэппинг библиотеки на адресное пространство процесса*

*Любое приложение состоит из секций (стека, данных и т.д.). Некоторые секции остаются общими, а некоторые секции создаются для каждого процесса.*

*Библиотека содержит точку входа – функция dllMain. Принимает параметры: хэндл – число, которое является номером объекта ядра ОС (когда мы создаем файл, процесс, поток возвращается хэндл). Это то число которое означает номер объекта внутри ядра ОС. И когда мы вызываем системный вызов и указываем хэндл, ядро понимает какой объект мы имеем ввиду. Так вот, длл когда загружается она тоже создает объект ядра ос.*

*Все что может использоваться между разными процессами, создается объект ядра ОС. Второй параметр это причина вызова (4 штуки).* **DLL\_PROCESS\_ATTACH -** когда DLL загружается в адресное пространство процесса

**DLL\_PROCESS\_DETACH -** когда DLL выгружается из адресного пространства процесса

**DLL\_THREAD\_ATTACH -** когда создается новый поток в процессе, к которому уже подключена DLL

**DLL\_THREAD\_DETACH-** когда поток завершает работу в процессе, к которому подключена DLL

*Понятное дело, что dllMain у каждого процесса свой.*

*LoadLibrary и FreeLibrary.*

*Функции нужно экспортировать.*

*2 способа экспорта функции:*

1. *extern “C” \_\_declspec(dllexport) int Sum(int x, int y) {return x + y}*

*extern “C” – указатель компилятору чтобы он не искажал имя функции Sum. Имя функции включал идентификатор и через спец знаки типы параметров.*

*Эта херня сделана чтобы в си плас плас была перегрузка функций, но нам будет неудобно вызывать функцию с таким идентификатором, поэтому мы говорим компилятору что нам нужен просто идентификатор Sum (т.к. в си перегрузки нет и имена не искажаются)*

*2)через .def файл. Вижла считывает этот файл, подает на вход компоновщика, и он указанные функции экспортирует*

*GetProcAddress(хэндл loadlibrary, “Имя функции”) – получение адреса функции*

*ДРУГОГО СПОСОБА ПОЛУЧИТЬ АДРЕС ФУНКЦИИ ИЗ ДЛЛ НЕТ. ТОЛЬКО ЧЕРЕЗ LOADLIBRARY И GETPROCADDRESS. И ЭТА ФУНКЦИЯ ДОЛЖНА БЫТЬ ЭКСПОРТИРУЕМА*

*Библитека импорта – статическая. Компонуется на этапе компоновки*

*#pragma comment(lib, “путь к .lib”)*

*Extern “C” int Sum(int,int);*

*В библиотеке импорта происходит loadLibrary, FreeLibrary, GetProcAddress. Просто все это обернуто в функцию (Sum). Т.е. внутри функции Sum есть getProcAddress и loadlibrary, извлекает эту функцию. Т.е. внутри статической библиотеки выполняется getProcAddress и loadlibrary. Просто все это скрыто.*

1. Спецификация COM: понятие позднего связывания программных модулей, COM-интерфейс, стандартные COM-интерфейсы, структура COM-клиента, структура COM/DLL-сервера, экспортируемые стандартные функции, регистрация COM/DLL-сервера.

*- для чего нужен COM?*

*Какие есть функции которые экспортируются. Для чего каждая нужна. Для чего фабрика классов. Как устроен ком компонент. Где создается и когда удаляется. Какие счетчики поддерживаются. Как регистрация осуществляется (в dll есть код который регистрирует). Ком – правило написания кода. Правило написания приложений которое было принято в майкрасофт.*

*Для чего нужен ком?*

*ЛК: COM – спецификация разработки приложения в виндоус. ОС виндоус придумали стандарт разработки приложений. Обеспечивает позднее связывание. Раньше технология называлась оле32. COM – модель ПО. Впринципе может применяться во всех ОС. Задача ком программирования – разработка специального ПО которое соответсвутет модели КОМ.*

*КОМ программирование – компонентное программирование.*

*У каждого компонента есть идентификатор (CLSID). Идентификатор построен по алгоритму GUID (128 битное число которое с очень низкой вероятностью может повториться).*

*В результате у нас есть ком объекты и ком экземпляры.*

*КОМ компонент и КОМ объект это если что одно и то же, это что-то вроде класса в ООП. А вот КОМ экземпляр это уже как объект у класса в ООП. Может быть выбрано 2 типа компонентов, либо будет находиться внутри длл, либо внутри exe. Т.е. в качестве капсулы для наших компонентов.*

*Многокомпонентная длл -–длл с несколькими ком компонентами. Ну и эту длл с компонентами называют ком сервером*

все константы и прототипы функций необходимые для поддержки COM-программирования становятся доступными с помощью **#include <objbase.h>**

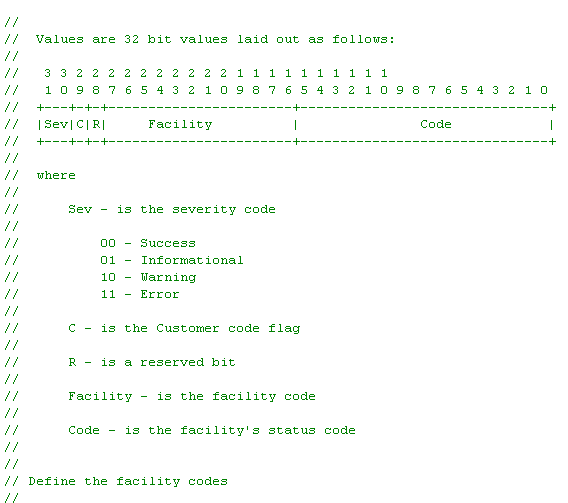
3 типа серверов: **CLSCTX\_INPROC\_SERVER (длл внутри процесса), CLSCTX\_LOCAL\_SERVER(**EXE-сервер за границами процесса, но та том же компьютере**), СLSCTX\_REMOTE\_SERVER (**EXE-сервер на удаленном компьютере**)**

**КОМ клиент – программный модуль который создает ком экземпляр и использует его. Он обращается к ком серверу, в рамках его создает экземпляр .** КОМ объекты могут быть клиентами, т.е. ком объект обращается к другому ком объекту**.**

1. при создании объекта посредником между COM-клиентом и COM-сервером выступает библиотека **OLE32.DLL** (библиотека импорта **OLE32.LIB**).



Все функции оле32 длл должны возвращать структуру HRESULT (4 байта, первые 2 бита тип сообщения, 1 бит – наш код возвращаем или код фрэймворка, facility – подсистема ос, код в рамках facility) (кроме 2 Addref и Release)

****

*Любой ком объект характеризуется набором интерфейсов и clsid. Чтобы мы могли добраться к функциям ком объекта, нам нужно это делать через интерфейс. Ком компонент содержит 1 или несколько интерфейсов. Каждый интерфейс имеет свой GUID.*

*Каждый интерфейс характеризуется CLSID и типом контейнера в котором он находится*

*Среди всех интерфейса есть 2 стандартных:* **IUnknown и IClassFactory. Объекты ком знают их идентификаторы, они в качестеве константы записаны в define. А еще все знают какие там методы вызываются, какие параметры передаются и что возвращается**

**Любой** *ком компонент обязан поддерживать* **IUnknown. Более того, каждый интерфейс должен быть унаследован от IUnknown**

1. **IUnknown:** имеет три метода: **QueryInterface** – получить адрес интерфейса по его идентификатору(первый параметр метода **QueryInterface это id, а второй это адрес, куда мы поместим результат функции**); **AddRef** (увеличить на 1 счетчик ссылок на интерфейс); **Release**(уменьшить счетчик ссылок на интерфейс).

*Когда мы ввытаскиваем интерфейс, мы должны отметить что мы его забрали. А когда он нам больше не нужен мы должны сообщить об этом. Внутри экземпляра компонента хранится счетчик количество выбранных интерфейсов. Т.е. если счетчик равен 0 то можно выгрузить экземпляр.*

*Все методы интерфейса должны поддерживать* **stdcall соглашение (**аргументы передаются через стек, справа налево, очистку стека производит **вызываемая** подпрограмма**)**

**COM-клиент**: для создания COM-объекта и работы с ним должен знать: 1)**CLSID объекта**; 2) **тип DLL-сервера** (контейнера); 3) **ID** **интерфейсов** объекта



CoInitialize – инициализация библиотеки OLE32

*CoCreateInstance(CSLID, NULL, SERVER\_TYPE, ID\_INTERFACE, (void\*\*)&ID\_INTERFACE) – создание экземпляра объекта ком*

*SUCCEEDDED – марок что все ок*

*Через полученный интерфейс мы можем вызывать методы интерфейса*

*CoFreeUnusedLibraries() – завершение работы с OLE32*



Соответствие между clsid и dll находится в реестре

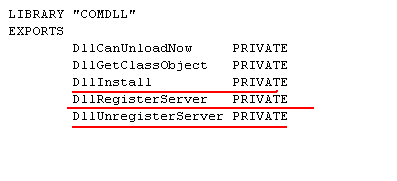
OLE32 как раз таки и берет информацию о clsid из реестра

Regsvr32 – регает/анргает com dll

Regsvr32 просто ищет функцию register/unregister в вашей dll и вызывает ее

Install – что-то еще попутно сделать с регистрацией/анрегистрацией

Dll com должен экспортировать след функции:



Писал чат:

HRESULT RegisterServer(

HMODULE hModule, // Дескриптор модуля DLL, используемый для получения пути к DLL

const CLSID& clsid, // Идентификатор класса (CLSID) компонента

const WCHAR\* szFriendlyName, // Читаемое имя компонента

const WCHAR\* szVerIndProgID, // Программный идентификатор версии (Version Independent ProgID)

const WCHAR\* szProgID // Программный идентификатор (ProgID)

);

* **hModule**: Дескриптор модуля DLL. Обычно это значение используется для получения пути к DLL, которая содержит COM компонент. Значение передается через функцию **LoadLibrary**.
* **clsid**: clsid
* **szFriendlyName**: Удобочитаемое имя компонента. Это строка, которая описывает компонент в более понятной форме, чем идентификатор CLSID.(чтобы в реестре было удобнее искать)
* **szVerIndProgID**: Программный идентификатор, независимый от версии (Version Independent ProgID). Это строка, которая используется для идентификации компонента без учета версии. Примером может быть **MyComponent**.
* **szProgID**: Программный идентификатор (ProgID). Это строка, которая включает версию компонента, например, **MyComponent.1**. Используется для регистрации конкретной версии компонента.

HRESULT UnregisterServer(

const CLSID& clsid, // Идентификатор класса (CLSID) компонента

const WCHAR\* szVerIndProgID, // Программный идентификатор версии (Version Independent ProgID)

const WCHAR\* szProgID // Программный идентификатор (ProgID)

);

* **clsid**: Идентификатор класса (CLSID) компонента. Это глобально уникальный идентификатор, используемый для идентификации COM компонента.
* **szVerIndProgID**: Программный идентификатор, независимый от версии (Version Independent ProgID). Используется для удаления всех записей, связанных с данным идентификатором.
* **szProgID**: Программный идентификатор (ProgID). Используется для удаления всех записей, связанных с конкретной версией компонента.

***DllCanUnloadNow*** *– можно ли выгрузить com сервер*

*Если счетчик ком экземпляров больше 0 или если сервер заблокирован (lockServer) то тоже нельзя выгрузить*

*Т.е. есть счетчик к-ва ком экземпляров, и счетчик ссылок на интерфейс*

***Счетчики:***

*Счетчик ссылок на интерфейс – можно ли выгрузить экземпляр*

*Счетчик к-ва ком экземпляров – можно ли выгрузить dll*

*Ну и счетчик lockServer*

*Любой экземпляр может заблокировать*

***DllGetClassObject*** *– вызывается через OLE32, вызывается по createInstance. Именно она создает экземпляр и через нее потом отдается интерфейс*

*DllGetClassObject(clsid, id\_iclassfactory)*

*Короче мы сюда передаем clsid объекта и iclassfactory и как бы говорим OLE32 “Верни мне указатель на фабрику классов, которая может создавать экземпляры объекта, с указанным clsid”*

***Фабрика классов*** *– экземпляр, который реализует интерфейс IClassFactory*

*Фабрика классов создает экземпляры на основе объектов*

*ДЛЯ КАЖДОГО ОБЪЕКТА ЕСТЬ СВОЯ ФАБРИКА КЛАССОВ*

*У IClassFactory есть методы: createInstance, lockServer*

*IClassFactory реализует интерфейс IUnknown*

*У CFactory тоже есть AddRef и Release*

*Кста, есть еще интерфейс IDispatch, но мы его не проходим. Его в офисе юзают*

1. Управление пользователями и группами пользователей в Windows: понятие дискреционной системы безопасности, типы Windows- пользователей, группы пользователей, возможности API управления пользователями и группами.

- ***как можно использовать API*** *Windows для авторизации в приложении?*

*Все что можно делать с апи и как это можно использовать(можно не писать никаких баз данных, ничего не придумывать а просто использовать то что уже есть в ОС). Можно создать свои собственные группы, можно делать аутентификацию, используя возможности ОС (если что в sql эта windows аутентификация она делается как раз через этот прикол)*

**Дискреционная система безопасности (Discretionary Access Control, DAC)** - это модель контроля доступа, которая определяет, какие пользователи или субъекты имеют право доступа к ресурсам системы.

**Дискреционное разграничение доступа к объектам** (Discretionary Access Control — DAC) характеризуется следующим набором свойств:

* все субъекты и объекты компьютерной системы должны быть однозначно идентифицированы;
* для любого объекта компьютерной системы определен пользователь-владелец;
* владелец объекта обладает правом определения прав доступа к объекту со стороны любых субъектов компьютерной системы;
* в компьютерной системе существует привилегированный пользователь, обладающий правом полного доступа к любому объекту (или правом становиться владельцем любого объекта).

Дискреционная система безопасности реализуется обычно в виде матрицы доступа, строки которой соответствуют субъектам компьютерной системы, а столбцы — ее объектам.

Элементы матрицы доступа определяют права доступа субъектов к объектам. В целях сокращения затрат памяти матрица доступа может задаваться в виде списков прав субъектов или в виде списков контроля доступа.

К достоинствам дискреционной системы безопасности относятся относительно простая реализация и хорошая изученность.

Недостатки. Прежде всего, к ним относится статичность разграничения доступа — права доступа к уже открытому субъектом объекту в дальнейшем не изменяются независимо от изменения состояния компьютерной системы.

При использовании дискреционной системы безопасности не существует возможности проверки, не приведет ли разрешение доступа к объекту для некоторого субъекта к нарушению безопасности информации в компьютерной системе.

Наконец, к недостаткам еще относится автоматическое назначение прав доступа субъектам.

В дискреционных системах есть понятия принципал. От лица которого исполняются все действия.

Юзеры могут быть принципалами. Всё, что выполняется, происходит от лица какого-то принципала.

API для справочника юзеров.

типы Windows-пользователей

Все учетные записи три типа:

администратор компьютера,

ограниченная учетная запись

гостевая учетная запись.

Пользователь с ограниченной учетной записью может выполнять операции со своим паролем (создание, изменение, удаление), изменять рисунок своей учетной записи, параметры настройки рабочего стола, а также просматривать файлы.

Учетная запись администратора позволяет выполнять следующие операции:

• Создание, удаление и редактирование учетных записей пользователей и своей;

• Операции со своим паролем;

• Установка и удаление программ и оборудования, редактирование их параметров и свойств;

• Чтение всех общих файлов;

• Внесение изменений в конфигурацию на уровне системы.

Что касается гостевой записи, то она формируется автоматически в процессе установки системы, и предназначена для сторонних пользователей, не имеющих на данном компьютере собственных учетных записей. Под учетной записью гостя нет доступа к файлам, папкам, параметрам и приложениям, которые защищены паролем. С помощью соответствующих ссылок можно включать/отключать учетную запись гостя.

**Группа ­–** совокупность пользователей с определёнными правами или **–** это набор учетных записей пользователей, которые объединены по какому-либо признаку. При этом отметим, что одна учетная запись пользователя может входить более чем в одну группу. Каждая группа имеет свою учетную запись и наделена своими правами и полномочиями. Эти права и полномочия передаются каждому члену группы.

**Группы пользователей**

* Администраторы. Неограниченный доступ.
* Операторы архива. Члены данной группы имеют права создания резервной копии даже тех объектов, к которым не имеют доступа.
* Опытные пользователи. Группа включена только для совместимости с предыдущими версиями
* Пользователи системного монитора. С помощью *Системного монитора* можно отследить использование различных ресурсов компьютером. А группа дает доступ к данному инструменту.
* Операторы настройки сети. Члены группы могут изменять параметры TCP/IP.
* Пользователи удаленного рабочего стола. Смогут входить в систему через удал. рабочий стол.
* Пользователи журналов производительности. Более полные права к *Системному монитору*.
* Криптографические операторы. Члены данной группы могут выполнять криптографические операции.
* Читатели журнала событий.

Привилегии можно назначать напрямую пользователю либо включать его в группу и назначать **группе привилегии**, тогда он будет владеть всеми этими привилегиями.

При авторизации каких-то операций, можно проверить принадлежность группы.

Встроенные пользователи: Администратор и др. (их нельзя удалить)

Встроенные группы: Все, Администраторы и др.

Пользователь наследует все свойства группы пользователей.

**Идентификация**: Это первый шаг, когда информационная система определяет, существует ли конкретный пользователь или нет. [Это делается с помощью идентификатора, который может быть логином, электронной почтой, номером телефона или другим признаком, который есть только у одного пользователя](https://skillbox.ru/media/code/identifikatsiya-autentifikatsiya-avtorizatsiya-chem-oni-razlichayutsya/)

1. [**Аутентификация**: Это второй шаг, когда пользователь вводит ключ (например, пароль или пин-код), подтверждая своё право на доступ к той или иной учётной записи и хранящейся в ней информации1](https://skillbox.ru/media/code/identifikatsiya-autentifikatsiya-avtorizatsiya-chem-oni-razlichayutsya/).
2. [**Авторизация**: Это последний шаг, который определяет, какие действия пользователь может выполнять после успешной аутентификации1](https://skillbox.ru/media/code/identifikatsiya-autentifikatsiya-avtorizatsiya-chem-oni-razlichayutsya/).

Субъекты безопасности – активные, те, что что-то делают (процессы, потоки)

Объекты – те, над кем выполняются действия (файлы и тд)

Возможности API управления пользователями и группами

**1. Имя текущего пользователя**

BOOL GetUserName

(

LPTSTR lpBuffer, // указатель на буфер

LPDWORD nSize // получает размер возвращает количество символов

);

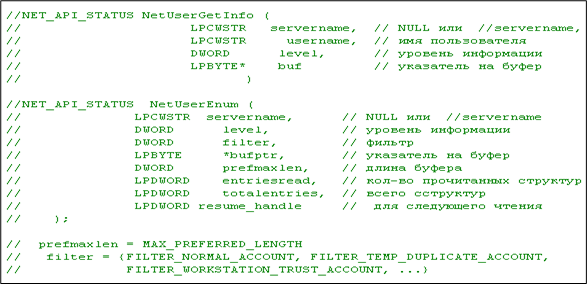
**2. Информация о пользователей**

****

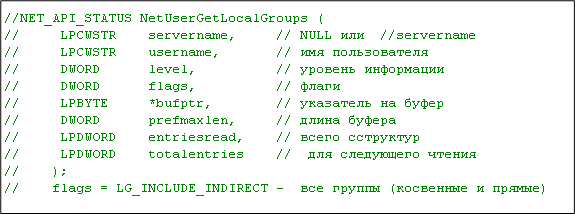
**3. Освободить память**

****

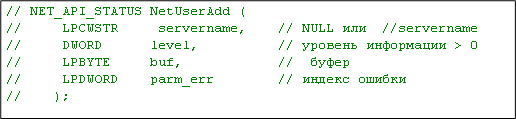
**4. Получить коллекцию пользователей**

****

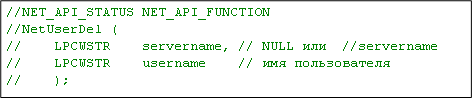
**5. Группы пользователя**

****

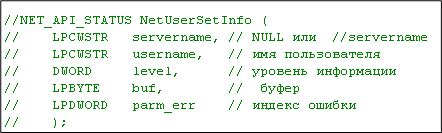
**6. Добавить пользователя**

****

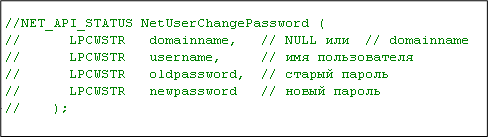
**7. Удалить пользователя**

****

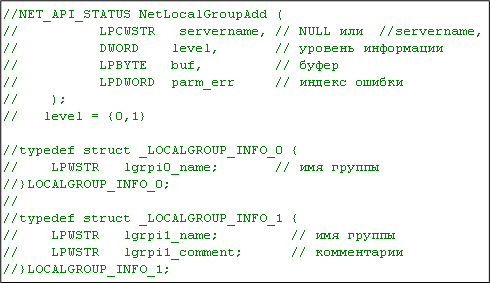
**8. Изменить информацию о пользователе**

****

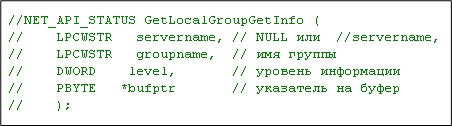
**9. Изменить пароль**

****

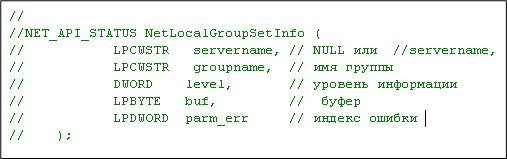
**10. Добавление группы**

****

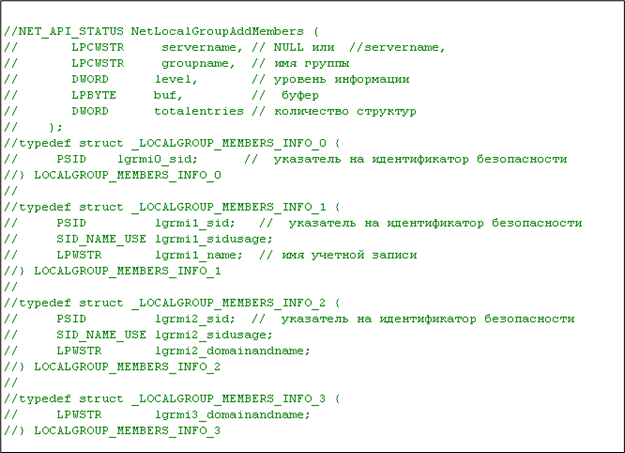
**11. Получить информацию о группе**

****

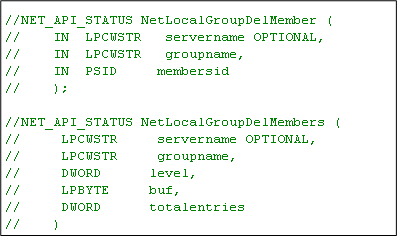
**12. Изменить информацию о группе**

****

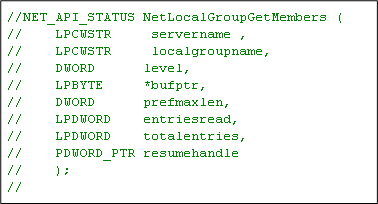
**13. Добавление членов локальной группы**

****

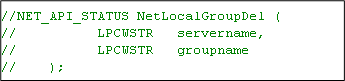
**14. Удаление членов группы**

****

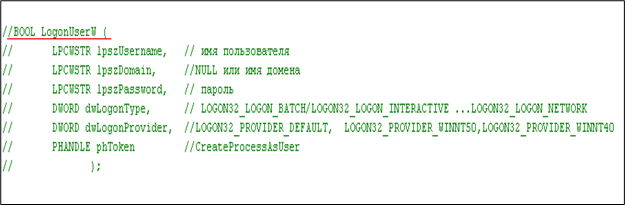
**15. Получить коллекцию членов группы**

****

**16. Удалить группу**

****

**17. Подключение пользователя**

****

*Дальше тоже чат:*

**Методы авторизации с использованием Windows API**

1. **Windows Integrated Authentication (WIA)**:
   * Это метод аутентификации, который использует учетные записи и группы Windows для управления доступом к ресурсам.
   * Полезен для приложений, работающих в доменной среде.
2. **API для управления учетными записями и группами пользователей**:
   * Использование функций Windows API для управления учетными записями пользователей и группами, проверки их прав и привилегий.
3. **Security Support Provider Interface (SSPI)**:
   * Это API Windows для реализации различных схем аутентификации, включая Kerberos, NTLM и другие.
4. **Token-based Authorization**:
   * Работа с токенами доступа, предоставляемыми системой после аутентификации пользователя.

### Пример использования Security Support Provider Interface (SSPI)

SSPI позволяет использовать различные протоколы аутентификации. Рассмотрим простой пример аутентификации с использованием NTLM.

#include <windows.h>

#include <sspi.h>

#include <stdio.h>

void AuthenticateWithSSPI()

{

CredHandle hCred;

TimeStamp tsExpiry;

SECURITY\_STATUS ss;

// Получение учетных данных по умолчанию

ss = AcquireCredentialsHandle(

NULL, // Имя пользователя

L"NTLM", // Поставщик безопасности

SECPKG\_CRED\_OUTBOUND, // Учетные данные для исходящих соединений

NULL, // Зарезервировано

NULL, // Данные для аутентификации

NULL, // Указатель на функцию

NULL, // Данные для функции

&hCred, // Дескриптор учетных данных

&tsExpiry); // Срок действия учетных данных

if (ss != SEC\_E\_OK)

{

printf("AcquireCredentialsHandle Error %u\n", ss);

return;

}

printf("Credentials acquired successfully\n");

// Здесь можно продолжить с вызовом InitializeSecurityContext и других функций SSPI для завершения аутентификации.

// Освобождение учетных данных

FreeCredentialHandle(&hCred);

}

int main()

{

AuthenticateWithSSPI();

return 0;

}

1. Структурная обработка ошибок в Windows: программное исключение, программные конструкции для обработки ошибок в Windows, фильтры, возможности API для структурной обработки ошибок, генерация ошибок, финальная обработка исключений.

* *зачем нужен SEH?*
* *когда надо применять SHE?*

*Для чего нужен сех. Почему есть трай кэтч, а есть сех. Потому что трай кэтч он привязан к системе программирования (языку). Он может не знать о некоторых исключениях которые происходят в операционной системе. Это языки платформонезависимые. И си ++ тоже старается делать платформонезависимые. А есть какие то особенности которые все же нужно обрабатывать в си ++ вот они и запихнули. Каким образом идет распространение? (есть спец код который приделывается к каждому приложению) (на самом деле main вызывается не напрямую а вызывается через спец код который в себе выдает такой трай, и внутри трая вызывается main функция)*

Обработка ошибок – механизм, кот. встраивается в ЯП, чтобы обозначить некоторые особенности ОС.

**Исключение** – событие в программе, произошедшее во время ее выполнения, в результате которого нормальное выполнение программы становится невозможным. Для дальнейшей работы приложения требуется либо восстановить ее рабочее состояние, либо аварийно ее завершить с очищением всех ресурсов этой программы. Для этого применяется механизм SEH.

**SEH (structured exception handling) –** является низкоуровневым механизмом операционной системой Windows в том смысле, что все ошибки (аппаратные и программные сбои, ошибки исполнения программы), возникающие при выполнении программ в Windows, обрабатываются именно по этой схеме. Все остальные способы обработки ошибок, предоставляемые языками программирования, в конце концов, сводятся к SEH.

SEH может отлавливать не только программные, но и аппаратные ошибки

SEH: Фрейм – блок кода, в котором может произойти исключение(внутри try). Код называется охраняемым кодом.

SEH: Обработчик исключения – блок программного кода, который обрабатывает исключение.

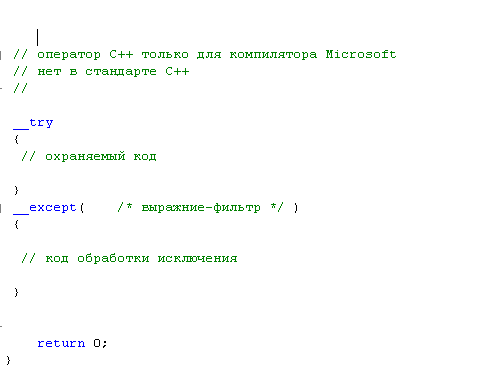
Исключения могут быть основаны на оборудовании или ПО. Структурированная обработка исключений полезна даже в том случае, когда приложения не могут полностью восстанавливаться после исключений. SEH позволяет отображать сведения об ошибках и захватывать внутреннее состояние приложения, чтобы помочь в диагностике проблемы.

**Программные конструкции для обработки ошибок**

Составной оператор после \_\_try предложения — *тело* или *защищенный* раздел.

\_\_except выражение также называется критерием *фильтра*. и в круглых скобках установлено соответствующее значение, называемое фильтром. Его значение определяет, как обрабатываются исключения. Обработчик задает действия, выполняемые при возникновении исключения во время выполнения раздела body. Выполнение происходит следующим образом:

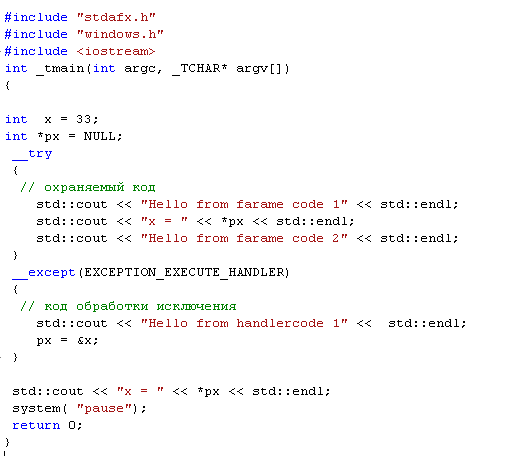
1. Сначала выполняется защищенный раздел \_\_try.
2. Если исключение при этом не возникает, выполнение переходит в инструкцию, стоящую после предложения \_\_except.
3. Если во время выполнения защищенного раздела возникает исключение или в любой подпрограмме вызывается защищенный раздел, \_\_except выражение вычисляется. Возможны три значения.
   1. EXCEPTION\_CONTINUE\_EXECUTION (-1) Исключение закрыто. Выполнение продолжается в точке, в которой возникло исключение.
   2. EXCEPTION\_CONTINUE\_SEARCH (0) исключение не распознано. Продолжайте выполнять поиск обработчика в стеке, сначала для содержащихся try-except инструкций, а затем для обработчиков со следующим высшим приоритетом.
   3. EXCEPTION\_EXECUTE\_HANDLER (1) распознано исключение. Передайте управление обработчику исключений, выполнив \_\_except составной оператор, а затем продолжайте выполнение после \_\_except блока.

****

нельзя goto в охраняемый код и в обработчик.

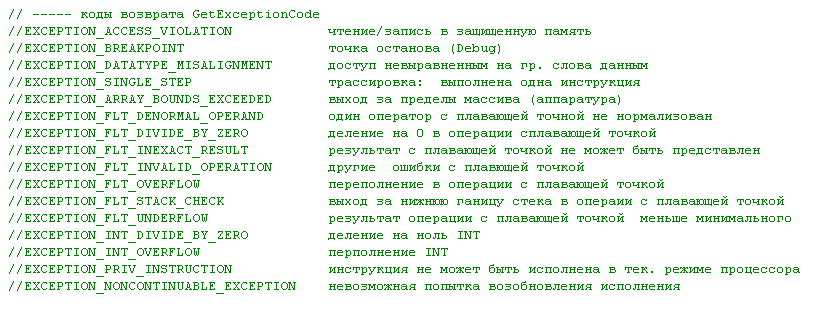
в выражении фильтра можно использовать две функции: GetExeptionCode, GetExceptionInformation.

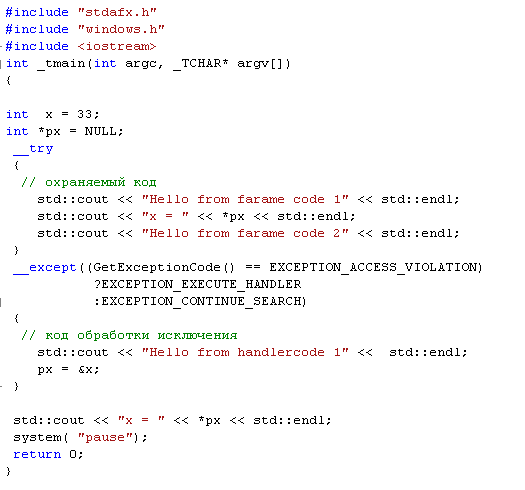
переменные, объявленные внутри {} – локальные

****

try/catch/throw ~ \_\_try/\_\_except: \_\_try/\_\_except разработан раньше, обрабатывает аппаратные ошибки.

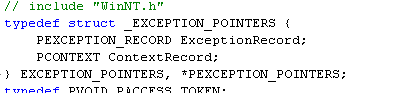
DWORD GetExceptionCode() – может использоваться только в выражении-фильтре (для определения дальнейших действий: обрабатывать, искать обработчик, вернуть управление в точку прерывания) или в блоке обработки исключения (для получения кода исключения).

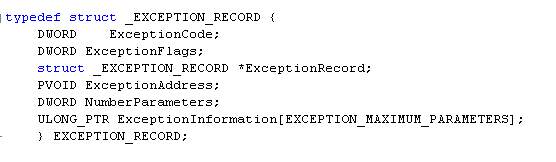
****

****

В функции фильтра нельзя вызывать GetExeptionCode или GetExceptionInformation, но можно использовать для инициализации параметров этой функции в выражении-фильтре. Если будет сделана попытка возобновить процесс выполнения программы, то в общем случае это не удается: один оператор С++, как правило, состоит из нескольких инструкций процессора, а возврат осуществляется к инструкции, что может привести к зацикливанию

LPEXCEPTION\_POINTERS GetExceptionInformation() – может быть использована только в выражении фильтра

****

****

ExceptionCode = GetExceptionCode()

ExceptionAddress – адрес инструкции

ExceptionFlag = 0 – возможно возобновление работы

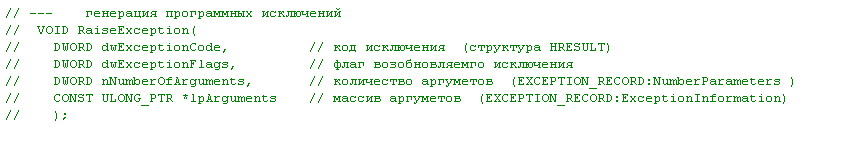
ExceptionFlag = EXEPTION\_NONCONTINUABLE – невозможно

\*ExceptionRecord = указатель на EXEPTION\_RECORD при вложенных исключениях

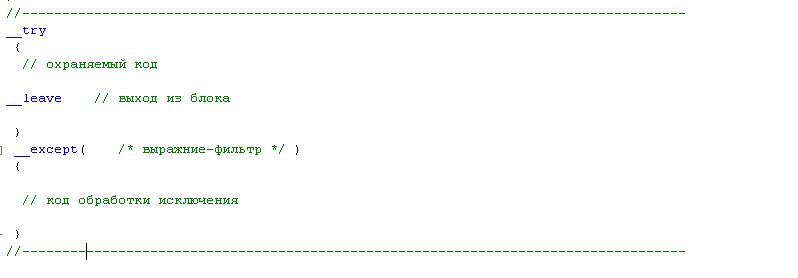
NumberParameters – количество параметров в ExcetptionInformation

ExcetptionInformation – массив 32-битных описателей исключения

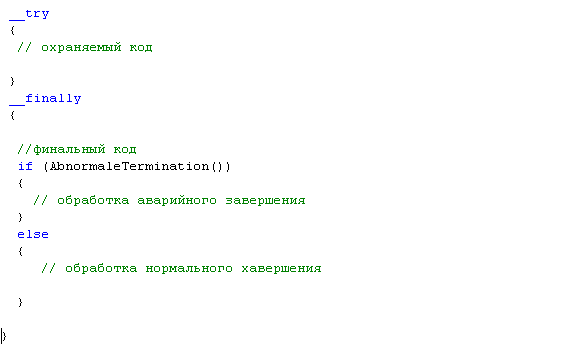
**SEH: Microsoft C++: генерация программных исключений**



SEH: Microsoft C++: передача управления

****

**SHE: Microsoft C++: финальная обработка исключений**

****

*Тут тоже чат:*

Исключения в Windows распространяются по стеку вызовов до тех пор, пока не будет найден подходящий обработчик исключений. Этот процесс включает в себя:

1. **Создание исключения**: Процессор или операционная система генерирует исключение.
2. **Поиск обработчика**: Система начинает поиск обработчика исключений, начиная с текущей функции и поднимаясь вверх по стеку вызовов.
3. **Выполнение обработчика**: Если подходящий обработчик найден, он выполняется. Если обработчик не найден, программа завершает свою работу.

#### Специальный код для каждого приложения

Каждое приложение в Windows может использовать SEH благодаря встроенной поддержке операционной системы и компилятора. Компилятор (например, MSVC) вставляет специальный код в программу для управления структурой SEH, что позволяет обрабатывать исключения на уровне языка C/C++.

Компилятор генерирует и вставляет инструкции для создания записей структурных обработчиков (SEH records) в стек вызовов. Эти записи используются для поиска и вызова соответствующих обработчиков исключений, когда они возникают.

1. Windows-консоль: определение, применение стандартных потоков для ввода/вывода в консоль, возможности API для управления консолью.

- *как можно применить API Console?*

*Что ты можешь сделать с консолью? Могу использовать несколько буферов 1 экран. Могу позиционировать. Могу менять курсор.*

*Ну и как можно применить консоль. С помощью этого api можно сделать собственный интерфейс. Разработать собственную cmd.*

**Windows-консоль (Интерфейс командной строки)** — разновидность текстового интерфейса между человеком и компьютером, в котором инструкции компьютеру даются в путём ввода с клавиатуры текстовых строк.

**Консоль —** это приложение, которое предоставляет службы ввода-вывода для приложений в символьном режиме.

**Консоль** – это три компонента: потоки ввода вывода, буфер и окно (Смелов на лекции)

Входной буфер консоли содержит очередь записей, которые описывают события ввода. События ввода подразделяются на следующие категории:

* + ввод с клавиатуры;
  + ввод с мыши;
  + изменение размеров окна;
  + изменение фокуса ввода;
  + события, связанные с меню.

Буфер экрана является двумерным массивом, элементы которого представляют собой записи типа:

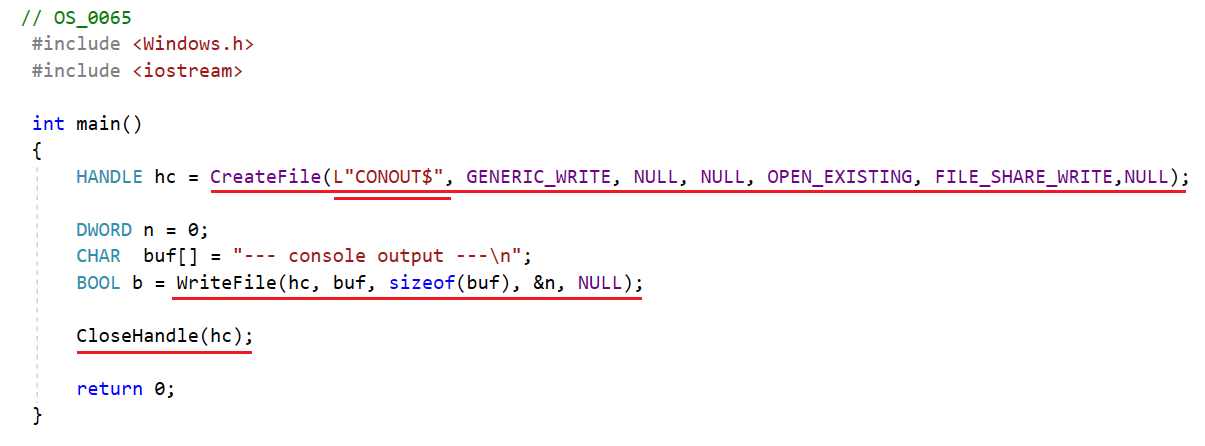
typedef struct \_CHAR\_INFO {  
union {  
WCHAR UnicodeChar;  
CHAR AsciiChar;  
} Char;  
WORD Attributes;  
} CHAR\_INFO, \*PCHAR\_INFO;

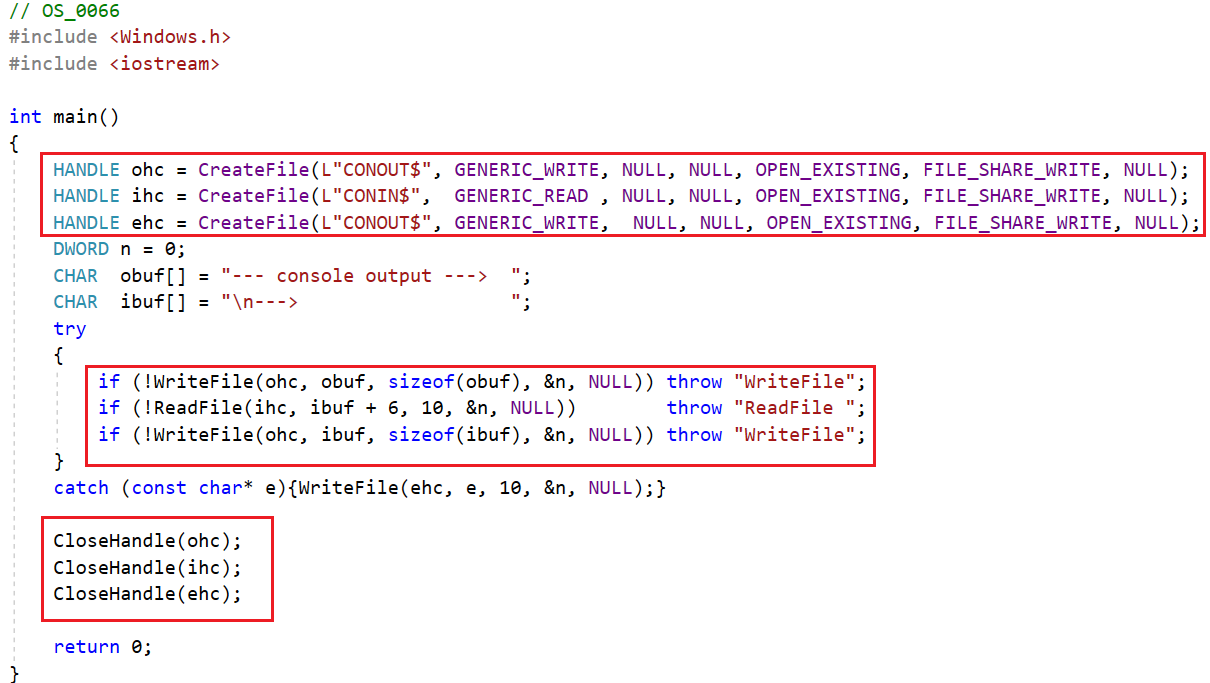
где объединение Char содержит символ, представленный в коде Unicode или ASCII, а поле Attributes определяет цвет фона и цвет текста, которыми выводятся символы на экран дисплея. Это значение может быть равно 0, что обозначает фон — черный, а цвет — белый, или любой комбинации из следующих констант:  
BACKGROUND\_BLUE — фон синий;  
BACKGROUND\_GREEN — фон зеленый;  
BACKGROUND\_RED — фон красный;  
BACKGROUND\_INTENSITY — фон яркий;  
FOREGROUND\_BLUE — текст синий;  
FOREGROUND\_GREEN — текст зеленый;  
FOREGROUND\_RED — текст красный;  
FOREGROUND\_INTENSITY — текст яркий.  
Цвет фона и цвет текста будем называть атрибутами текста. Сделаем несколько замечаний относительно использования этих констант. Цвет фона и цвет текста выбираются как комбинация базовых цветов синего, зеленого и красного. То есть в этом случае используется цветовая модель RGB. Можно подсчитать, что всего существует семь возможных комбинаций из трех цветов.  
Белый цвет определяется комбинацией всех трех цветов. Если сюда добавить черный цвет, который определяется как побитовое отрицание белого цвета, то всего существует восемь возможных вариантов, как для цвета фона, так и для цвета текста.

Консоль — это приложение, которое предоставляет службы ввода-вывода для приложений в символьном режиме.

Консоль состоит из входного буфера и одного или нескольких буферов экрана. Входной буфер содержит очередь входных записей, каждая из которых содержит сведения о событии ввода. Очередь ввода всегда включает события ключа и нажатия клавиши. Он также может включать события мыши (перемещения указателя и нажатия кнопки и выпуски) и события, в течение которых действия пользователя влияют на размер активного буфера экрана. Буфер экрана — это двумерный массив символьных и цветовых данных для вывода в окне консоли. Консоль может совместно использоваться любым количеством процессов.

Стандартные потоки:

****

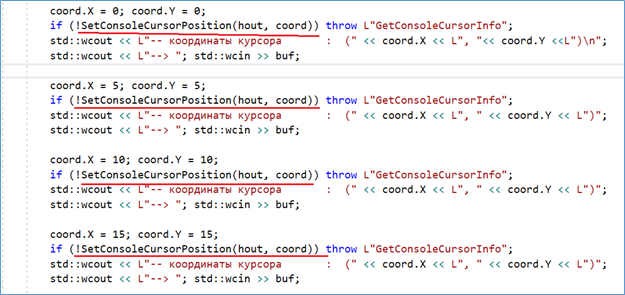
****

Функция [CreateFile](https://docs.microsoft.com/ru-ru/windows/win32/api/fileapi/nf-fileapi-createfilea) позволяет процессу получить дескриптор для входного буфера консоли и активного буфера экрана, даже если STDIN и STDOUT были перенаправлены. Чтобы открыть дескриптор для входного буфера консоли, укажите значение CONIN$ при вызове CreateFile. Укажите значение CONOUT$ при вызове CreateFile, чтобы открыть дескриптор для активного буфера экрана консоли. CreateFile позволяет указать доступ только для чтения и записи в возвращаемом дескрипторе.

Console: параметры консоли



**Console: позиции курсора**

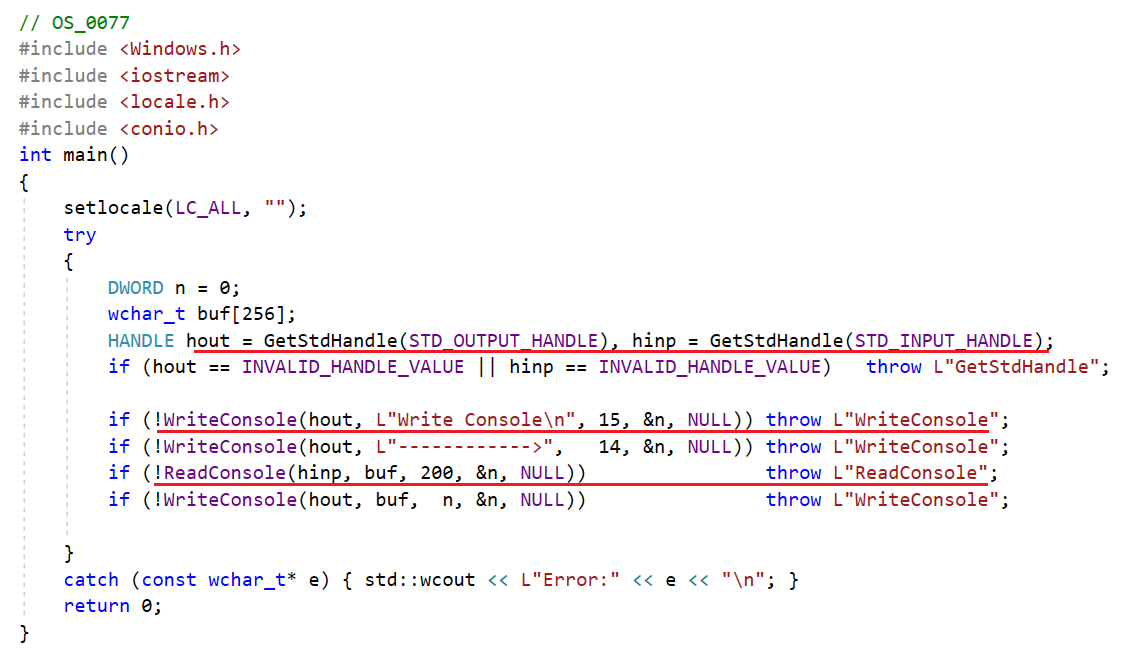


**Console: установка атрибутов консоли**



**Высокоуровневый ввод/вывод WriteConsole, ReadConsole**

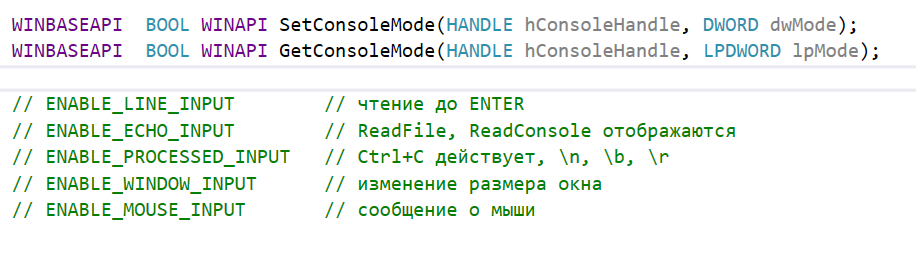
ReadFile, WriteFile – аналогичны WriteConsole, ReadConsole



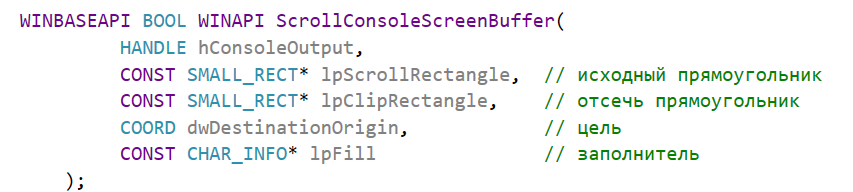
**Console: ввод низкого уровня ReadConsoleInput**



**режимы работы консоли**



**прокрутка буфера экрана**



SetConsoleTitle задаёт заголовок для текущего окна консоли.

WriteConsole записывает строку символов в буфер экрана консоли, начиная с текущего положения курсора.

ReadConsole считывает входные символы из буфера ввода консоли и удаляет его из буфера.

GetCurrentConsoleFont извлекает сведения о текущем шрифте консоли.

FlushConsoleInputBuffer очищает входной буфер консоли.

AllocConsole выделяет новую консоль для вызывающего процесса.

AddConsoleAlias определяет псевдоним консоли для указанного исполняемого файла.

*Чет чат еще написал:*

**Что ты можешь сделать с консолью?**

Консоль в Windows предоставляет обширные возможности для взаимодействия с операционной системой и выполнения различных задач. Вот ключевые возможности, которые можно использовать с консолью:

1. **Ввод и вывод данных**:
   * **Чтение данных** с клавиатуры и **вывод информации** на экран.
   * Перенаправление ввода и вывода в файлы и другие устройства.
2. **Управление текстом и атрибутами экрана**:
   * Изменение **цвета текста** и **фона**.
   * Управление стилями текста, такими как мигающий текст или яркость.
3. **Управление курсором**:
   * **Перемещение курсора** по экрану.
   * **Получение и установка позиции курсора**.
4. **Работа с буфером экрана**:
   * Чтение и запись данных в **буфер экрана**.
   * Создание, изменение размеров и переключение между буферами экрана.
5. **Обработка ввода**:
   * Управление режимами ввода (например, переход в режим без эха).
   * Обработка **клавиатурных и мышиных событий**.
6. **Управление консольным окном**:
   * Изменение размеров консольного окна.
   * Управление заголовком окна.
7. **Управление процессами и командами**:
   * Выполнение команд и скриптов.
   * Управление системными службами и процессами.

**Как можно применить консоль?**

Консоль в Windows применяется для множества задач и ситуаций. Вот несколько примеров её применения и почему именно консоль является предпочтительным инструментом:

1. **Автоматизация задач**:
   * **Автоматизация повторяющихся задач** через скрипты и командные файлы (batch files).
   * **Пример**: Ежедневное создание резервных копий данных с помощью командного файла.
2. **Администрирование системы**:
   * **Удаленное управление и администрирование** серверов и рабочих станций.
   * **Пример**: Запуск диагностики системы или установка обновлений на удалённом сервере через SSH.
3. **Разработка и тестирование**:
   * Быстрая и **эффективная отладка** и тестирование программ без необходимости сложных графических интерфейсов.
   * **Пример**: Тестирование новых функций программного обеспечения с использованием командной строки.
4. **Управление файлами и каталогами**:
   * Быстрое и **эффективное управление файлами** и папками (копирование, перемещение, удаление).
   * **Пример**: Массовое переименование файлов в каталоге с помощью командных файлов.
5. **Мониторинг и диагностика**:
   * **Мониторинг системных ресурсов** и диагностика проблем.
   * **Пример**: Использование команды **tasklist** для просмотра запущенных процессов и их использования памяти.
6. **Низкое потребление ресурсов**:
   * Консольные приложения потребляют меньше ресурсов по сравнению с графическими интерфейсами.
   * **Пример**: Запуск скриптов на сервере без графического интерфейса для экономии ресурсов.
7. **Обработка и фильтрация данных**:
   * **Фильтрация и обработка данных** на лету.
   * **Пример**: Использование команд **find** и **sort** для обработки текстовых данных.
8. **Создание и управление пользовательскими интерфейсами**:
   * Разработка простых **текстовых интерфейсов** для взаимодействия с пользователем.
   * **Пример**: Создание меню для выбора опций в командном файле.
9. Windows-сервисы: определение, назначение, применение, API.

- *в каких случаях надо использовать сервисы?*

1. Асинхронные операции ввода вывода: понятие асинхронной операции ввода/вывода, особенности программирования асинхронного ввода/вывода.

*- в каких случаях надо использовать асинхронный ввод и вывод?*

1. Порты завершения ввода/вывода: назначение, применение, API.

*- в каких случаях надо использовать порты завершения ввода и вывода?*

1. Платформа Docker: архитектура, назначение, принципы устройства, файловая система UFS, контейнеры, образы, основные команды.
2. Платформа Docker: процесс разработки Image.
3. Платформа Docker: процесс разработки серверного приложения.

Доцент каф. ИСиТ В.В. Смелов