# Windows HOOK. Назначение и особенности использования.

Хуками называются перехватчики системных событий. Они позволяют программно подменять обработчик этих событий.

**Назначение:**

Хуки необходимы для считывания нажатий и корректной обработки их.

Хуки используются при любом нажатии и событии в системе Windows. Хуки используются, например, в Visual Stydio при создании события onClick.

Любое событие в системе может вызвать другое при обработке хуком.

**Особенности использования:**

За установку хука отвечает функция SetWindowsHookEx Она возвращает Указатель на структуру HHOOK.

Для того чтобы интерпретировать события нажатия виртуальные клавиши используется функция TranslateMessage.

Для отправки сообщения в обработчик хука используется функция DispatchMessage.

Функция UnhookWindowsHookEx(hHook); Освобождает дескриптор хука.

# Актуальность синхронизации потоков при параллельной обработке данных

Поток – последовательный набор инструкций, некий объект исполняемого процесса.Актуальность синхронизации потоков при параллельной обработке данных заключается в том, что она исключает возможность неправильной поведении программы. Например, когда несколько потоков получают доступ к одному ресурсу, значение может быть записано или прочитано не вовремя.

Когда потоки пытаются использовать один ресурс одновременно, программа может работать не корректно. Например, при одновременно увеличении переменной на 1 разными потоками, некоторые могут не выполнить необходимые действия, так как его выполняет другой поток. Значение может быть записано или прочитано раньше времени.

Существует много способов синхронизации и потоков, главным принципом является запрет параллельного доступа к общим ресурсам. Например, при помощи создания критической секции.

# Библиотеки DLL. Назначение и использование DLL-библиотек

Не все функции, описанные в проекте, единовременно могут быть использованы, но они занимают пространство оперативной памяти и это сказывается на объёме исполняемого файла.

Если одни и те же функции необходимо использовать в разных проектах, то код приходится дублировать. Причём не всегда копирование кода решает эту проблему.

Данные проблемы как раз-таки решается за счет подключения динамических библиотек.

Динамическая библиотека содержит код функций, который может быть использован в нескольких программах одновременно.

Использование DLL-библиотек позволяет экономить оперативную память, так она подключается к проекту только в момент ее вызова, и отключается после того как пропала необходимость в ней. Для работы с динамической библиотекой, ее необходимо подключить с помощью функции, далее импортировать функции и в окончании отключить библиотеку.

DLL (динамические) подключаются к проекту и отключаются в нужный момент времени. Таким образом экономится ОП.

Существует несколько типов DLL-библиотек, в зависимости от того, как они используются:

* Статически связанные DLL (Static Link Library) - это DLL, которые статически связаны с программой и загружаются в память при запуске программы. Этот тип DLL используется редко, так как он занимает больше памяти.
* Динамически связанные DLL (Dynamic Link Library) - это DLL, которые динамически связаны с программой и загружаются в память по требованию. Этот тип DLL чаще всего используется, так как он экономит память.
* Системные DLL - это DLL, которые содержат системные функции иресурсы, необходимые для работы операционной системы. Системные DLL являются частью ОС и используются множеством программ.
* Специальные DLL - это DLL, которые содержат специализированные функции и ресурсы, необходимые для работы определенных программ. Специальные DLL могут создаваться разработчиками для решения конкретных задач.

Использование DLL-библиотек может улучшить производительность, упростить разработку и увеличить надежность программы. Однако также существует ряд ограничений и рисков, связанных с использованием DLL, таких как необходимость учитывать версии DLL и совместимость с различными ОС.

# Многопоточный режим работы компьютера. преимущества использования многопоточности

В основе парадигмы программирования лежит понятие процесс.

Процесс можно понимать, как некий объект для исполняемого файла. Один и тот же исполняемый файл может быть запущен несколько раз (при этом создаётся несколько процессов).

Вся информация о процессе хранится в оперативной памяти.

Процесс характеризуется неким набором информации. За работу процессора отвечают потоки.

Поток рассматривается как последовательный набор инструкций, который выполняется на процессоре. Каждый процесс имеет хотя бы один поток. Данный поток называется основным.

Процессы могут находиться в одном из следующих состояний:

• Выполняется. Основному потоку выделено процессорное время. Количество выполняемых процессов одновременно и не может быть больше, чем потоков, которые поддерживает процессор.

• Состояние готовности к выполнению. Процессу предоставлены все ресурсы, кроме процессорного времени.

• Состояние ожидания. Процессу предоставлены не все ресурсы (Например, идёт ввод или вывод данных).

За время своего существования один и тот же процесс может многократно менять свои состояния.

Место процесса в очереди определяется его приоритетом.

Приоритет процесса может поменять приоритет (в том числе и создаться) в одном из следующих случаев:

• По команде пользователя.

• При выборе из очереди планировщиком операционной системы

• По таймеру системному

• По инициативе другого процесса

При создании процесса создаётся его дескриптор, который является структурой, содержащий всю необходимую информацию о процессе.

Основной поток в процессе может также порождать и вспомогательные потоки.

Они нужны для параллельного выполнения операций.

Плюсы Минусы

Переключение между выполняющимися процессами потребляет заметную долю временных и других ресурсов ОС, а в случаях, аналогичных многопроцессному поиску (grepMP, программа 6.1), все процессы заняты выполнением одной и той же программы. Организация параллельной обработки файла с помощью потоков в контексте единственного процесса позволяет снизить общие накладные расходы системы. Поскольку потоки разделяют общую память и другие ресурсы, принадлежащие одному процессу, существует вероятность того, что один поток может случайно изменить данные, относящиеся к другому потоку.

С использованием только однопоточных процессов трудно организовать простое и эффективное управление несколькими параллельно выполняющимися задачами, взаимодействующими между собой, в таких, например, случаях, как ожидание и обработка пользовательского ввода, ожидание ввода из файла или сети и выполнение вычислений. При определенных обстоятельствах вместо улучшения производительности может наблюдаться ее резкое ухудшение.

Улучшенная реакция приложения - любая программа, содержащая много не зависящих друг от друга действий, может быть перепроектирована так, чтобы каждое действие выполнялось в отдельном потоке. Например, пользователь многопоточного интерфейса не должен ждать завершения одной задачи, чтобы начать выполнение другой.

Создание многопоточной программы требует очень тщательной разработки. Вероятность появления незначительных временных сбоев или ошибок, вызванных нечаянным совместным использованием переменных, в такой программе весьма значительна. Алан Кокс (Alan Сох, всеми уважаемый гуру Linux) сказал, что потоки равнозначны умению "выстрелить в обе собственные ноги одновременно".

Улучшенная структура программы - некоторые программы более эффективно представляются в виде нескольких независимых или полуавтономных единиц, чем в виде единой монолитной программы. Многопоточные программы легче адаптировать к изменениям требований пользователя. Отладка многопоточной программы гораздо труднее, чем отладка одного потока исполнения, поскольку взаимосвязи потоков очень трудно контролировать.

Переключение между потоками требует от операционной системы гораздо меньше усилий, чем переключение между процессами. Программа, в которой громоздкие вычисления разделены на две части, и эти две части выполняются как отдельные потоки, необязательно будет работать быстрее на машине с одним процессором, если только вычисление не позволяет выполнять обе ее части одновременно и у машины, на которой выполняется программа, нет многоядерного процессора для поддержки истинной многопоточности.

# Особенности использования строкового типа данных. Массивы строк

Строка - это последовательность ASCII или UNICODE символов.

Строковый тип данных в C как таковой отсутствует, а в качестве строк используются обычные массивы символов.

Исторически сложилось два представления формата строк:

1. формат ANSI;

2. cтроки с завершающим нулем (используется в C).

Формат ANSI устанавливает, что значением первой позиции в строке является ее длина, а затем следуют сами символы строки.

В строках с завершающим нулем, значащие символы строки указываются с первой позиции, а признаком завершения строки является значение ноль.

Так как строки на языке С являются массивами символов, то к любому символу строки можно обратиться по его индексу. Для этого используется синтаксис обращения к элементу массива, поэтому первый символ в строке имеет индекс ноль.

Объявление массивов строк в языке С также возможно. Для этого используются двумерные массивы символов, что имеет следующий синтаксис:

Char имя[количество][длина];

Первым размером матрицы указывается количество строк в массиве, а вторым – максимальная (с учетом завершающего нуля) длина каждой строки. Например, объявление массива из пяти строк максимальной длиной 30 значащих символов будет иметь вид:

char strs[5][31];

Все функции используют заголовочный файл string.h. Все библиотечные функции, предназначенные для работы со строками, можно разделить на три группы:

1. ввод и вывод строк;

2. преобразование строк;

3. обработка строк.

Для ввода и вывода строковой информации можно использовать функции форматированного ввода и вывода (printf и scanf). Для этого в строке формата при вводе или выводе строковой переменной необходимо указать спецификатор типа %s.

В С для преобразования строк, содержащих числа, в численные значения в библиотеке stdlib.h предусмотрен следующий набор функций:

double atof(const char \*string); // преобразование строки в число типа double

int atoi(const char \*string); // преобразование строки в число типа int

long int atol(const char \*string); // преобразование строки в число типа long int

long long int atoll(const char \*string); // преобразование строки в число типа long long int.

Вообще говоря, символ (символ) занимает 1 байт или 2 байта, мы говорим, что 1-байтовый символ является ANSI, он может использоваться для представления английских букв, а 2 байта, которые мы называем Unicode, могут представлять Все языки в мире – префикс L.

VC ++ использует встроенные типы данных char и wchar\_t для представления символов ANSI и Unicode соответственно.

char\*Замена:LPSTR - это (Длинный) указатель на строку.

const char\*Замена:LPCSTR

WCHAR\*Замена:LPWSTR

const WCHAR\*Замена:LPCWSTR

TCHAR\*Замена:LPTSTR

const TCHAR\*Замена:LPCTSTR

'L' означает wchar\_t, который, в отличие от нормального символа, требует 16 бит хранения, а не 8 бит. Вот пример:

"A" = 41

"ABC" = 41 42 43

L"A" = 00 41

L"ABC" = 00 41 00 42 00 43

A wchar\_t в два раза больше, чем просто char. В ежедневном использовании вам не нужно использовать wchar\_t, но если вы используете windows.h, вам это понадобится.

# Особенности настройки проекта и использования синтаксиса языка программирования при написании DLL-файла

Он настраивается также, как и проект winapi, Ну помимо этого ещё необходимо указать В настройках проекта на вкладке общие Тип конфигурации динамическая библиотека.

Точка входа у библиотеки DLL тоже своя особенная:

BOOL WINAPI DllMain(мне кажется это достаточно)

(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; }

Все экспортируемые из DLL функции должны иметь специальные соглашения о вызовах \_cdecl. (Дело в том, что приложение winapi по умолчанию имеет соглашения вызовах \_\_stdcall).

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 мы её импортировать или экспортировать не сможем (Но просто писать функции в dll мы можем).

# Особенности разработки программ в WinAPI (основные отличия от программ, не использущих API)

Windows API спроектирован для использования в языке Си для написания прикладных программ, предназначенных для работы под управлением операционной системы MS Windows. Работа через Windows API — это наиболее близкий к операционной системе способ взаимодействия с ней из прикладных программ.

Windows API представляет собой множество функций, структур данных и числовых констант, следующих соглашениям языка Си.

В WinAPI отличается соглашение о вызовах.

Все языки программирования, способные вызывать такие функции и оперировать такими типами данных в программах, исполняемых в среде Windows, могут пользоваться этим API. Пример: C++, C#, Pascal, Visual Basic.

Для облегчения переноса на другие платформы программ, написанных с опорой на Windows API, сделана библиотека Wine.

Функция запуска приложений Windows

Функция называется WinMain. Принимает 4 параметра.

WINAPI WinMain(HINSTANCE hlnstance. // дескриптор, присваиваемый запущенному приложению

HINSTANCE hPrevInstance, // для совместимости с winl6. в Win32 не используется

LPSTR lpCmdLine. // указатель на командною строку, если приложение так запущено

int nCmdShow); // значение, которое может быть передано в функцию Show Window ()

Перед запуском приложений Windows необходимо особым образом настроить проект.

Также при работе с API системы Windows необходимо подключать заголовочный файл windows.h.

С помощью WinAPI мы можем работать с системными окнами Windows.

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

При разработке программ для работы с системными событиями использования мыши и клавиатуры следует учитывать следующие особенности:

* Необходимость обработки различных событий, связанных с мышью и клавиатурой, таких как нажатие кнопки, отпускание кнопки, перемещение указателя мыши и т.д.
* Необходимость реагирования на различные события в реальном времени, чтобы обеспечить плавное и удобное использование программы.
* Необходимость учитывать различные настройки и опции, связанные с мышью и клавиатурой, такие как скорость курсора, размер шрифта и т.д.
* Необходимость учитывать различные особенности работы с системными событиями в разных операционных системах.
* Необходимость обеспечить безопасность и надежность работы с системными событиями, чтобы избежать возможных ошибок и сбоев в работе программы

Для разработки программ, работающих с системными событиями использования мыши и клавиатуры, часто используются специализированные библиотеки и фреймворки, такие как WinAPI, MFC, Qt и т.д. Они предоставляют готовые функции и классы для работы с системными событиями, что упрощает разработку и увеличивает надежность программы.

# Подходы к обработке исключений в программах WinAPI, написанных на си

использование возвращаемых значений функций и последующая их проверка, анализ и т.д.;

• использование исключений и обработка исключительных ситуаций.

Исключение возникает только в случае непредвиденной ошибки времени выполнения и обрабатывает все остальное с кодами ошибок и результатов — напрямую и близко к источнику сбоя. Таким образом, при вызове исключения вы будете знать, что его причина заключается в ошибке в вашем коде или в состоянии исключения системы.

Рассмотрим сценарий доступа к реестру Windows. Если вашему приложению не удается прочитать значение из реестра, следует ожидать ошибку, которую необходимо корректно обработать. Не вызывайте исключение, а верните значение bool или enum, указывающее, что значение не было прочитано и, возможно, почему это произошло. Тогда как ошибка записи значения в реестр скорее всего указывает на более серьезную проблему, которую невозможно обработать в вашем приложении. В таком случае не следует продолжать работу приложения, поэтому создание исключения, которое создает отчет об ошибках, — этой самый быстрый способ предотвращения дальнейших проблем.

Обработка ошибок на основе анализа возвращаемого значения производится вне метода, в котором возникла ошибка. Т.е. для использования такого подхода вы должны быть уверены, что метод хоть что-то вернёт, какая бы ошибка внутри него не возникла. Велика вероятность, что вы об этом не узнаете: если возникнет ошибка, ваша программа может быть просто остановлена системой.

Механизм обработки ошибок на основе исключений работает внутри метода и, если обработчик исключения в этом методе не найден, то распространяется на следующий метод в стеке (тот, из которого был вызван метод, в котором произошло исключение) и т.д. И только если обработчик исключения так и не будет найден, система аварийно завершит выполнение вашей программы.

# Понятие “системное программирование”. предназначение и специфика системного программирования

Системой называется отношение двух множеств: множество элементов и множество связей между этими элементами.

Программы называется набор инструкций, понятных для конечного исполнителя.

Программирование - Это процесс написания программного кода.

Программный код - это зашифрованная версия программы (набор инструкций, написанный на языке, понятном постановщику задачи).

Системные программисты пишут программные коды для функционирования системы.

Операционная система выступает посредником между пользователем и аппаратной частью компьютера.

Основная отличительная черта системного программирования по сравнению с прикладным программированием заключается в том, что результатом прикладного является выпуск программного обеспечения, предлагающего определенные услуги пользователям (текстовый процессор). Результатом системного программирования является выпуск программного обеспечения, предлагающего сервисы по взаимодействию с аппаратным обеспечением (дефрагментация жёсткого диска), что подразумевает сильную зависимость таких программ от аппаратной части.

# Предназначение и особенности использования директив препроцессора и заголовочных файлов.

Серьёзные программные продукты имеют достаточно большое количество строчек кода. Если весь программный код помещается в одном файле, то его восприятие становится затруднительным.

Поэтому логически завершенные фрагменты программного кода рекомендуется помещать в отдельные файлы.

Так как множество строк кода приходится повторять в разных файлах, такие как объявление глобальных переменных и указание прототипов функций, их обычно выносят в заголовочные файлы.

Такие файлы называются файлами заголовков.

Файлы заголовков имеет расширение .h и добавляется в проекте стандартным способом (с помощью #include).

Пользовательские файлы заголовков указывается в двойных кавычках.

Файлы заголовков рекомендуется использовать для объявления объектов и не рекомендуется для их инициализации.

Препроцессор - это специальная программа, которая осуществляет алгоритмические действия перед компиляцией основного кода.

Команды для препроцессора называются директивами.

Директивы начинаются с символа #, в конце строки директивой “;” можно не ставить.

Список основных директив:

#include. Вставляет содержимое из текстового файла в то место, где она написана

#define. Имеет три основных применения:

• Инициализация параметров (Задание флагов). Используется преимущественно для условной компиляции

• Задание констант. Используется в качестве альтернативы глобальным переменным

• Задание макроопределений (макросов)

#undef. Отменяют задание параметра

#ifdef - Условие компиляции, если определённый параметр задан

#ifndef - Условие компиляции, если определённый параметр не задан

#if - Инициализация условной компиляции. далее необходимо ввести условие, используя другие директивы препроцессора.

#elif - Директивы для создания вложенных условий.

#else - Ветка при ложности всех условий. Её нельзя ставить выше чем #elif

#error - Внесение искусственной ошибки для компиляции

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

Операционная система выступает посредником между пользователем и аппаратной частью компьютера. Где каждый процесс — это отдельно взятая запущенная программа.

Windows выступает «посредником» между программой и устройствами компьютера.

Процесс можно понимать, как некий объект для исполняемого файла. Один и тот же исполняемый файл может быть запущен несколько раз (при этом создаётся несколько процессов).

Вся информация о процессе хранится в оперативной памяти.

Процесс характеризуется неким набором информации. За работу процессор отвечают потоки.

Поток рассматривается как последовательный набор инструкций, Который выполняется на процессоре. Каждый процесс имеет хотя бы один поток. Данный поток называется основным.

Место процесса в очереди определяется его приоритетом.

Дескриптор (HANDLE) — это идентификатор объекта, который создается не вашей программой, а операционной системой или сторонней библиотекой.

При создании процесса создаётся также его дескриптор, который является структурой, содержащий всю необходимую информацию о процессе.

# Принципы функционирования системного программного обеспечения (на основе анализ отличий от других типов ПО)

Системное программное обеспечение — это комплекс программ, которые обеспечивают управление компонентами компьютерной системы, такими как процессор, оперативная память, устройства ввода-вывода, сетевое оборудование, выступая как «межслойный интерфейс», с одной стороны которого аппаратура, а с другой — приложения пользователя.

В отличие от программ, которые взаимодействуют непосредственно с пользователем (прикладное программное обеспечение), системное не решает конкретные практические задачи, а лишь обеспечивает работу других программ, предоставляя им сервисные функции, абстрагирующие детали аппаратной и микропрограммной реализации вычислительной системы, управляет аппаратными ресурсами вычислительной системы.

Оно разрабатывается так, чтобы компьютер мог эффективно выполнять прикладные программы.

Также к таким программам относятся драйвера, которые настраивают определённую аппаратную часть компьютера (Например, драйвера на видеокарту, мышку и так далее).

Главной особенностью функционирования системного программного обеспечения как драйвер можно назвать то что у него отсутствует графический интерфейс взаимодействия с пользователем, поскольку данная программа предназначена только для того, чтобы сообщить компьютеру как нужно обращаться с этим устройством и не требует никакого взаимодействия с пользователем после установки для корректной работы.

Не менее важной особенностью системного программного обеспечения является то что данное программное обеспечение пишется на языке низкого уровня. Из этого следует то что системное программное обеспечение управляет ресурсами компьютером напрямую и имеет наибольшие возможности управления компьютером и его составляющими (ОЗУ, сетевая карта), по сравнению с прикладными программами.

Также из этого следует то что что системное программное обеспечение работает в разы быстрее поскольку отсутствуют некоторые уровни абстракции, которые присутствуют в языках высокого уровня, которые предназначены для упрощения написания программ для решения прикладных задач.

# Назначение реестра Windows. Особенности использования системного реестра программами

Формально реестр представляет из себя базу данных.

Элементами этой базы данных являются параметры. Организация параметров представляет из себя древовидную иерархию.

Чаще всего реестр используется для хранения параметров операционной системы или прикладных программ.

Реестр задумывался как некое общее хранилище настроек. До этого программы хранили свои настройки в конфигурационных файлах. Использование конфигурационных файлов имела ряд недостатков:

• Не было централизованного хранилища этих файлов.

• Проблема защиты файлов:

− Текстовые файлы можно было удалить

− Можно было изменить содержимое текстовых файлов по-своему

Для открытия ключа реестра используется функция RegOpenKey. Она возвращает числовое значение (которое интерпретируется как код).

Функция RegCreateKey открывает ключ, если он есть или создаёт его.

Для того, чтобы создать параметр используется функция RegSetValue.

Для того чтобы получить значение параметра используется функция RegGetValue.

Для удаления параметра используется функция RegDeleteValue.

Для освобождения памяти под дескрипторы ключа используется функция RegCloseKey(hKey).

Вообще механизм его работы достаточно прост. При установке программы (или изменении какого-либо параметра Windows в любом из меню настроек) система сама разыскивает нужные параметры и вносит коррективы в одно из значений реестра. Программно можно изменить некоторые параметры, например, шрифт, используемый в приложении.

Хранение паролей для прикладных программ (практически нет никаких шансов получить какой-либо пароль с помощью простого просмотра данных реестра. Пароли, в подавляющем большинстве случаев, хранятся в зашифрованном виде, и для их дешифрации потребуется специальное программное обеспечение).

# Системный буфер обмена. Особенности взаимодействия программного обеспечения с системным буфером обмена

По сути буфер обмена можно рассматривать как канал или глобальный указатель, к которому имеют доступ все прикладные программы.

Однако при работе с буфером следует учитывать и его недостатки:

• Буфер обмена только один в системе и любое приложение может его перезаписать. Поэтому нет гарантии целостности получения данных

• Информация не может быть приватной (Во сколько буфер обмена доступен всем)

Буфер обмена позволяет хранить данные различных типов. Однако при работе с ним программы учитывают этот контекст. В зависимости от формата данных в буфере с ним могут взаимодействовать те или иные приложения.

Буфер обмена нужно скорее воспринимать как указатель, а не как переменную. У буфера нет своей заранее выделенной области. Поэтому память выделяется в самой программе.

Буфер обмена предоставляет набор возможностей для временного (не дискового) хранения информации.

Приложение-источник может копировать данные в буфер обмена в одном из предопределенных или пользовательских форматов. Буфер самостоятельно управляет выделением памяти и размещением переданных ему данных. После того как данные попадут в буфер обмена, любое приложение сможет получить к ним доступ, определить тип данных и при желании скопировать их.

Так как буфер обмена только один, все приложения должны использовать его совместно. Но совместное использование неизбежно связано с возможностью конфликтов. Предположим, что приложение А записало в буфер обмена растровое изображение, а затем приложение Б записало в него блок текстовых данных. Поскольку приложение Б вполне закономерно начинает с того, что очищает буфер обмена, картинка, записанная приложением А. удаляется. Если приложение В. Для которого была предназначена эта картинка, уже успело ее прочитать, то все в порядке. Если же приложение В не скопировало изображение до того, как приложение Б записало в буфер обмена текст, картинка пропадает.

Управление глобальными блоками памяти, которые содержат данные, помещенные в буфер обмена, осуществляется с помощью флагов выделения памяти. При записи информации в буфер программа выделяет блок памяти с помощью функции GlobalAlloc() и флага GHND (определенного как GMEM MOVABLE | GMEM ZEROINIT).

Обычно при закрытии исходного приложения выделенная ему глобальная область памяти удаляется (освобождается) операционной системой. Но если приложение вызовет функцию SetClipboardData с указанием дескриптора глобального блока памяти. Windows возьмет этот блок в свою собственность, точнее, в собственность буфера обмена, изменив флаги выделения.

Принадлежность глобального блока памяти назначается функцией GlobalRealloc().

После этого вызова выделенный блок памяти не принадлежит исходному приложению и доступен только через буфер обмена с помощью функции GetClipboardData(). Эта функция предоставляет вызвавшему ее приложению временный доступ к данным, записанным в буфере обмена, передавая программе дескриптор глобального блока памяти. Однако принадлежность блока данных по-прежнему сохраняется за буфером обмена, а не передается приложению. Следовательно, данные, записанные в буфере обмена, могут быть удалены только путем вызова функции EmptyClipboard.

Доступ к буферу обмена в каждый момент возможен только одной программой. Это позволяет предотвратить конфликты между приложениями.

Прежде чем приложение начнет читать, записывать или удалять содержимое буфера обмена, необходимо запросить доступ к нему с помощью функции OpenClipboard(). Эта функция возвращает значение TRUE, если буфер открыт и доступ к нему разрешен, и значение FALSE, если доступ к буферу запрещен по той причине, что в данный момент право доступа принадлежит другому приложению.

Закончив работу с буфером обмена, приложение должно вызвать функцию CloseClipboard. которая делает буфер доступным для других программ. Следует отметить, что каждый вызов функции OpenClipboard всегда должен сопровождаться вызовом функции CloseClipboard.

Функция TransferToClipboard начинается с запроса на право доступа к буферу обмена. Затем в буфер копируется одиночный блок памяти. Наконец, функция закрывает буфер обмена, освобождая его для доступа другим приложениям.

Прежде чем начинать чтение информации из буфера обмена, необходимо определить, какого типа данные содержатся в нем. Поскольку для различных типов данных необходимо применять разные операции, приложение должно заранее знать, что именно ему придется читать, и подготовиться к определенному виду обработки. В частности, вы можете запросить данные определенного типа и посмотреть на результат. Однако такой подход не слишком элегантен, да и эффективность его не очень высока.

Более эффективный способ анализа содержимого буфера обмена заключается в использованииАРІ-функции IsClipboardFonnatAvailable() или EnumClipboardFormats().

Функция IsClipboardFonnatAvailableO возвращает булево значение, которое сообщает, содержит ли буфер обмена данные нужного формата.

Функция EnumClipboardFormats проверяет наличие данных всех возможных форматов. При первом вызове функции с параметром NULL oнa возвращает информацию о первом доступном формате. При каждом последующем вызове возвращаются сведения о других форматах. Таким образом, для получения списка форматов можно воспользоваться следующим алгоритмом:

wFormat = NULL;

OpenCli pboa rd( hwnd);

while(wFormat = EnumClipboardFormals(wFomiat))

{...код обработки различных форматов...}

CloscClipboardO;

Следует отметить, что на операции, которые могут выполняться с буфером обмена, налагается несколько ограничений:

1. Прежде чем копировать данные в буфер обмена, необходимо вызвать функцию EmptyClipboard(). предназначенную для удаления текущего содержимого. Сам факт доступа к буферу обмена еще не означает, что будет получен контроль над его содержимым. Фу нкция Empty Clipboard() позволяет стать владельцем буфера и одновременно удалить его текущее содержимое.

2. Любое приложение может получить доступ к содержимому буфера обмена, но лишь владелец буфера, т. е. приложение, вызвавшее функцию Empty Clipboard(). имеет право записывать в него данные. Поскольку владельцем буфера обмена может быть только одно приложение, данные, записанные предыдущим владельцем, удаляются, даже если таковым было это же самое приложение.

3. Хотя в буфер обмена может быть записано несколько блоков данных, передавать их нужно в течение одной операции. Буфер обмена нельзя открыть, записать в него данные, закрыть, а затем снова открыть и добавить другие данные, не удалив предыдущий фрагмент.

4. В буфере обмена одновременно может находиться только по одному элементу данных каждого типа. Это объясняется простой причиной: не существует способа разделения нескольких элементов данных одного и того же типа. Однако при наличии данных нескольких типов приложение, получившее доступ к буферу обмена, может запросить только один элемент, несколько элементов или все элементы. В этом случае данные каждого типа должны запрашиваться отдельно.

Буфер обмена может многократно открываться для запроса других элементов данных или повторного запроса того же элемента. Но в целом, запрашивая фрагмент данных из буфера обмена. лучше всего создать его локальную копию, а не повторять один п тот же запрос снова и снова.

Кроме того, не забывайте: нет никакой гарантии того, что при следующем запросе тот же самый элемент данных останется неизменным, поскольку буфер обмена является общедоступным.

# Структура системного реестра. Типы данных в системном реестре.

HKEY CLASSES ROOT— хранится информация о зарегистрированных классах, расширениях документов;

HKEY CURRENT USER — хранится информация о текущей пользовательской конфигурации, внешнем виде рабочего стола, сетевых настройках;

HKEY LOCAL MACHINE — хранится информация о системной и аппаратной конфигурации;

HKEY USERS — хранится информация обо всех зарегистрированных пользователях;

HKEY\_CURRENT\_CONFIG — текущая аппаратная конфигурация.

Крупные ветки называются ульями.

Более мелкие ветки называются ключами реестра.

Каждый ключ может содержать внутри себя другие ключи или параметры.

Параметры также имеют свои определенные типы:

• Строковый параметр. Содержит последовательность символов в определенной кодировке. Его идентификатор в системе REG\_SZ

• Двоичный параметр. По сути содержит массив байт. Его идентификатор в системе REG\_BINARY

• Машинное слово для 32-битных систем. Его идентификатор в системе REG\_DWORD

• Машинное слово для 64 разрядной системы. Его идентификатор в системе REG\_QWORD

• Мультистроковый параметр. Другими словами, массив строк. Его идентификатор в системе REG\_MULTI\_SZ

• Расширяемый строковый параметр. По сути представляет из себя строку переменной длины. REG\_EXPAND\_SZ

# Структуры данных. актуальность использования структур. Передача структуры как параметра.

Структура — это, некое объединение различных переменных (даже с разными типами данных), которому можно присвоить имя.

Например, можно объединить данные об объекте Дом: город, улица, количество квартир и т.д. в одной структуре.

Структура также статический располагается в памяти, поэтому на неё тоже можно создать указатель.

Обращение к полю структуры через указатель осуществляется с помощью оператора Стрелка ->

Как и любой другой объект, структура может использоваться в качестве параметра функции и также может быть возвращаемым объектом функции.

При использовании структур в качестве параметров в функции следует учитывать, что при вызове функции для структуры также, как и для параметров типа int или char, выделяется память, в которую помещаются значения элементов структуры. То есть структура в функцию передается по значению, а это значит, что переданную в функцию структуру мы изменить не можем.

Если мы действительно хотим изменить структуру в функции, то надо передавать не саму структуру, а указатель на нее.

В итоге любое изменение значений по этому указателю приведет к изменению самой структуры. И результаты программы будут уже отличаться.

# Типы данных в ОС Windows. Назначение и использование системных диалоговых окон.

Беззнаковые целые числа представляются в виде последовательности битов в диапазоне от 0 до 2n-1, где n-количество занимаемых битов.

Знаковые целые числа представляются в диапазоне -2n-1…+2n-1-1. При этом старший бит данного отводится под знак числа (0 соответствует положительному числу, 1 – отрицательному).

• Тип BYTE обозначает 8-разрядное беззнаковое символьное значение. Тип byte — это знаковый 8-битовый тип. Его диапазон — от -128 до 127. Он лучше всего подходит для хранения произвольного потока байтов, загружаемого из сети или из файла.

• Тип WORD — 16-разрядное беззнаковое короткое целое.

• Тип DWORD — беззнаковое длинное целое.

• Тип UINT — беззнаковое 32-разрядное целое.

• Тип LONG эквивалентен типу long. Тип long предназначен для представления 64-битовых чисел со знаком. Его диапазон допустимых значений достаточно велик даже для таких задач, как подсчет числа атомов во вселенной.

• 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 (Дескриптор - это указатель или индекс, к которому не привязан видимый тип). Это просто число, определяющее некоторый ресурс.

Дескриптор выступает в качестве идентификатора определённого ресурса.

# Указатель как тип данных. Особенности работы с указателями

Указатель представляет из себя переменную, значением которой является адрес другой переменной.

Под указатель тоже выделяется место в оперативной памяти. Количество байт зависит от разрядности программы.

Переменные однозначно идентифицируется двумя параметрами: Это её адрес (Номер 1 ячейки памяти) и количество занимаемых ячеек.

К указателям применима операции сложения и вычитания. Они интерпретируются как сдвиги на определённое количество ячеек памяти. Шаг равен количество байтов, на которые ссылается указатель.

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

Фактически массив является указателем на выделенную область в памяти (количество элементов \* на размер элемента(тип данных)).

Имя массива является указателем на первый элемент. А объём памяти равен суммарному объему Всех элементов с учетом их типа.

В Си можно создавать динамические массивы. За динамическое выделение памяти отвечает функция malloc(), calloс(),realoс();

malloc(), В качестве аргумента данная функция принимает количество байт, которые нужно выделить.

Calloс()В качестве аргумента данная функция принимает количество элементов, и размер одного элемента(зануляет все выделенные ячейки памяти).

Realoс() изменяет ранее заданный размер области памяти с сохранением данных.

За освобождение памяти отвечает функция free(). В качестве аргумента функция принимает указатель.

Функция sizeof() принимает в качестве аргументов идентификатор типа, а возвращается количество байт, которое он занимает.

# Линейный односвязный список. Особенности создания и примеры использования.

Линейный однонаправленный список — это структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей. Каждый элемент списка – структура, в которой есть поле с адресом на другие элементы списка, а другая часть полей – данные.

Особенностью создания является то, что, так как это структура, то сектора в памяти выделяются не подряд. С использованием динамических структур можно проще вносить изменения в середину списка, так как при этом используются только соседние элементы.

Особенность создания заключается в том, что элемент не считается завершенным пока он не имеет связи со следующим элементом.

Примеры использования разнообразны так, как то, какие данные хранить в структуре выбирает программист. Наиболее удобной структура будет если необходимо поменять элементы местами или удалить элемент из середины без необходимости сдвига остальных элементов.

# Указатель на функцию. Назначение и примеры использования.

В языке программирования C функция имеет адрес и может иметь указатель. Указатель на функцию представляет собой выражение или переменную, которые используются для представления адреса функции. Указатель на функцию содержит адрес первого байта в памяти, по которому располагается выполняемый код функции.

Самым распространенным указателем на функцию является ее имя. С помощью имени функции мы можем вызывать ее и получать результат ее работы.

Под функцию также, как и под переменную выделяется память. Соответственно можно создать указатель на ту область памяти, в которой находится эта функция.

Если на уровне среды программирования или непосредственно в Си-программах имеет место элемент динамического связывания, то с уверенностью можно сказать, что в том или ином виде (явно или неявно) используется указатель на функцию:

• DLL, а также все возможные виды динамической загрузки внутреннего (исполняемого) программного кода;

Указатели широко используются в C в основных целях:

• передача функций в другие функции

• для итерации элементов в массивах или других структурах данных.

# Файл как тип данных. Использование файлов в работе программного обеспечения.

В программе могут быть использоваться файлы для ввода и вывода. Они находятся в файле <stdio.h>. В работе ПО файлы можно открыть, закрыть, прочитать и записать. Переменная типа файл представляет собой указатель на файловый поток. За открытие файла отвечает функция fopen. Существует несколько режимов доступа к файлу, например, только для чтения, только для записи, для обновления.

Файл можно читать и в файл можно записать. Для взаимодействия с ним нужно создать переменную – указатель на файловый поток, главные параметры, имя файла и пусть к нему и режим доступа. Если правильно распарсить, то в файле можно хранить числовые, текстовые данные. Также данные можно записать в структуру с определенными полями. Важно следить за размером буфера и соблюдать кодировки.

Файловый ввод-вывод находится также в заголовочном файле <stdio.h>

Типовыми функциями для работы с файлами являются следующие:

• Создание указателя на файл (Открытие файла)

• закрытие файла

• вывод информации из файла (чтение)

• Запись информации в файл

Для открытия файла в системе C необходимо объявить переменную типа файл (Точнее указатель на файловый поток)

За открытие файла отвечает функция fopen.

Данная функция принимает два параметра:

1. Имя файла (путь к нему)

2. Режим доступа к файлу (имеется в виду чтение или запись)

Список режимов:

r Чтение. Файл должен существовать.

w Запись нового файла. Если файл с таким именем уже существует, то его содержимое будет потеряно.

a Запись в конец файла. Операции позиционирования (fseek, fsetpos, frewind) игнорируются. Файл создаётся, если не существовал.

r+ Чтение и обновление. Можно как читать, так и писать. Файл должен существовать.

w+ Запись и обновление. Создаётся новый файл. Если файл с таким именем уже существует, то его содержимое будет потеряно. Можно как писать, так и читать.

a+ Запись в конец и обновление. Операции позиционирования работают только для чтения, для записи игнорируются. Если файл не существовал, то будет создан новый.

Если необходимо открыть файл в бинарном режиме, то в конец строки добавляется буква b, например, “rb”, “wb”, “ab”, или, для смешанного режима “ab+”, “wb+”, “ab+”. Вместо b можно добавлять букву t, тогда файл будет открываться в текстовом режиме.

Функция для чтения данных из файла:

За это 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)

Файлы в работе программного обеспечения необходимы как места временного хранения информации или передачи информации пользователю.

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

За создание дескриптора файла отвечает функция CreateFile.

HANDLE CreateFile(

LPCTSTR lpFileName, // Указатель на имя файла (устройства)

DWORD dwDesiredAccess, //Параметры доступа

DWORD dwShareMode, //Разделяемый доступ

LPSECURITY\_ATTRIBUTES lpSecurityAttributes, //безопасность

DWORD dwCreationDistribution,// Описание

DWORD dwFlagsAndAttributes, // Атрибуты файла

HANDLE hTemplateFile // Файл шаблона);

Запись данных в файл:

BOOL WriteFile(HANDLE hFile, //собственно указатель на файл

LPVOID lpBuffer, // указатель на буфер - откуда записываем данные в файл

DWORD nNumberOfBytesToWrite, //объем записываемых данных

LPDWORD lpNumberOfBytesWrite, //фактический размер записанных данных

LPOVERLAPPED lpOverlapped // флаг режима доступа к файлу: асинхронный(FILE\_FLAG\_OVERLAPPED)

//или синхронный(NULL)

конкретный пример записи в файл:

LPCSTR MyString = "Hello world";//буфер для записи (что записываем)

DWORD d = 0;

//функция записи в файл

WriteFile(hFile,//дескриптор открытого файла

MyString,//указываем буфер для записи

strlen(MyString), //указываем, сколько байт мы хотим записать

&d,//передаем указатель на DWORD

NULL//синхронный режим записи);

Функция для чтения данных из файла:

За это 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).

пример кода:

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

DWORD sizeBuffer = 521;//объем буфера

    LPSTR str = malloc(sizeBuffer+1);//куда считывать

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

    str[d] = '\0';

Пример со структурой Overlapped:

 OVERLAPPED olf = { 0 }; //Структура, в которой задана позиция в файле

    DWORD sizeBuffer = 512;//объем буфера

    LPSTR str = malloc(sizeBuffer+1);//куда считывать

    ReadFile(hFile, str, sizeBuffer, &d, &olf);

    free(str);

    olf.Offset = 0;//задаем смещение (позицию в файле)

    LPSTR str1 = malloc(d + 1);

    ReadFile(hFile, str1, d, &d1, &olf);

    str1[d1] = '\0';

Имеется два типа синхронизации ввода - вывода файлов: синхронный ввод - вывод файла и асинхронный.

При синхронном вводе – выводе файла поток запускает операцию ввода/вывода и немедленно вводит ждущееся состояние до тех пор, пока, запрос ввода-вывода не завершит работу. Поток, выполняющий асинхронный ввод - вывод файла, отправляет запрос на ввод-вывод данных ядру. Если запрос принят ядром, поток продолжает обрабатывать другое задание до тех пор, пока ядро не подаст сигналы потоку, что операция ввода/вывода полностью завершилась. Тогда поток прерывает работу со своим текущим заданием и обрабатывает данные от операции ввода/вывода по мере необходимости. Асинхронные операции позволяют выполнять ресурсоемкие операции ввода-вывода без блокировки основного потока.

В ситуациях, когда ожидается запрос на ввод-вывод, который займет большое количество времени, такое как обновление или резервное копирование большой базы данных, асинхронный ввод - вывод как правило - хороший способ оптимизировать эффективность обработки. Однако, для относительно быстрых операций ввода/вывода, непроизводительные издержки обработки запросов ядра на ввод-вывод и сигналов ядра могут сделать асинхронный ввод - вывод менее выгодным, особенно если должны делаться много быстрых операций ввода/вывода. В этом случае, синхронный ввод - вывод будет лучше.

Процесс открывает файл для асинхронного ввода - вывода при его вызове в CreateFile, устанавливая флажок FILE\_FLAG\_OVERLAPPED в параметре dwFlagsAndAttributes. Если FILE\_FLAG\_OVERLAPPED не определяется, файл открыт для синхронного ввода - вывода (I/O). Когда файл был открыт для асинхронного ввода - вывода (I/O), указатель на структуру OVERLAPPED передается при вызове в ReadFile и WriteFile. Структура не передается при вызовах в ReadFile и WriteFile, выполняя синхронный ввод - вывод.

# Анонимные и именованные каналы. Назначение и использование.

Канал (англ. Pipe) - область виртуального пространства, которое может быть использовано для совместного доступа различными процессами.

Однако в силу этого канал не может храниться отдельно от какого-то процесса (По факту дескриптор канала является глобальным указателем). Поэтому информация о нём будет очищена если завершить процесс, который его создал.

Будем называть процесс, который создаёт канал - сервером, а процессы, которые подключаются к каналу - клиентами.

Каналы имеют несколько разновидностей:

• Симплексные или дуплексные

o Симплексный это однонаправленные. Например, сервер только записывает данные, а Клиенты только читают их. Или клиенты только пишут, а сервер только читает.

o Дуплексные это когда и клиент, и сервер может 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 полем дескриптора имеют также имя в виде строки. Причём если дескриптор каждый раз разный, то имя является константой.

Имя является сетевым.

Сервер создаёт именованный канал с помощью функции CreateNamedPipe():

Она имеет следующую сигнатуру (мне кажется не обязательно это):

Сетевое имя канала

доступ (симплексный или дуплексный)

тип, режимы чтения и ожидания

максимальное число клиентов

размер выходного буфера, байты

размер входного буфера/байты

время паузы, миллисекунды (ожидание подключения)

структура безопасности

Проверить статус подключения клиента к серверу можно с помощью функции ConnectNamedPipe();

Для того чтобы установить подключение, на стороне клиента Должна быть вызвана функция SetNamedPipeHandleState();

Она имеет следующие аргументы:

• Дескриптор канала

• режим подключения

• максимальное количество пользователей

• Время ожидания

Пример: BOOL isSuccess = SetNamedPipeHandleState(hNamedPipe,&dwMode,NULL,NULL);

Если клиент подключился к серверу, ConnectNamedPipe функция возвращает значение True иначе возвращает false;

Если сервер не отвечает, то функция SetNamedPipeHandleState возвращает FALSE.

Далее вся логика функционирования именного канала настраивается исходя из контекста задачи.

Анонимные каналы.

В качестве их идентификатора используется дескрипторы на чтение или на запись.

Пример функции для создания анонимного канала

BOOL CreatePipe (PHANDLE pliRead. // переменная для дескриптора чтения (входной канал)

PHANDLE phWrite. // переменная для дескриптора записи (выходной канал)

LPSECURITY\_ATTRIBUTES Ipsa. // привилегии доступа

DWORD dwPipeSize); // размер буфера канала (0 по умолчанию)

Из недостатков такого способа следует отметить, что дескриптор канала сервер должен передать клиенту по какому-то другому пути.

# Создание процессов. Функция запуска. Аргументы командной строки

Процесс – это любая запущенная программа. Создать процесс можно с помощью функции CreateProcess. Она имеет 10 полей, как правило, многим из них присваивается значение по умолчанию.

Функция запуска называется WinMain. Она принимает 4 параметра, отвечающих за дескриптор, совместимость с Win16, указатель на командную строку и значение, которые может быть передано в функцию Show Windows().

Аргументы командной строки – строковые аргументы, для обмена данными при создании нового процесса (входные данные для программы).

Когда мы запускаем исполняемый файл, мы создаем процесс. Вся информация о котором хранится в ОП. Передача данных между процессами очень важна. Например, по каким-либо критериям можно запускать системные утилиты. Функцией запуска является функция main, которая по хорошему счету должна что-то принимать и возвращать на этом и строится методика создания процессов. Если программа завершилась с кодом 1 (например, проверка скорости чтения диска), запускается и выполняется программа по очистке кэша и временных файлов, если 0, то программа не запускается.

Main принимает (int args, char\* argv[])

char\* argv[] – массив строк

int args – количество аргументов

Если через командную строку выполнить исполняемый файл и написать произвольную строку, можно получить эту строку в программе, если вывести argv[].

# Ассемблерные вставки. Актуальность использования ассемблерных вставок. Технология создания ассемблерной вставки в языке си.

В программировании, ассемблерной вставкой называют возможность компилятора встраивать низкоуровневый код, написанный на ассемблере.

Для облегчения разработки применялся следующий подход: большая часть кода пишется на языке высокого уровня, и только участки, для которых критична производительность, либо требующие обращения непосредственно к аппаратным ресурсам компьютера, пишутся на ассемблере.

Использование ассемблерных вставок может преследовать следующие цели:

Оптимизация

Системные вызовы:  Языки программирования высокого уровня редко предоставляют прямую возможность делать системные вызовы, для этих целей используется ассемблерный код

