**Лекция 2**

**Обработка ошибок в Windows**

При реализации ПО надо учитывать, что любая ф-я WinAPI может быть обработана неверно, что для приложения может иметь последствия. Если вызывается ф-я WinAPI, она проверяет переданные параметры, а потом –выполняет свою работу. Параметры передаются стандартно. Передача параметров в ф-ю:

В ф-ю передается указатель на адрес памяти, где хранится параметр (даже если он простого типа). Потом ОС (в частности, транслятор языка) определяет размер параметра, его кодировку и тип. Если данные параметры не соответствует тому, что требует ф-я, происходит их изменение следующим образом: в зависимости от разрядности процессора происходит приведение типа (как расширяющее, так и сужающее), а также изменяется разрядность сетки.

Повторить двоичную СС, способы перевода

Если в ф-ю передается недопустимый параметр и ее нельзя выполнить еще по какой-то причине, ф-я возвращает значение, говорящее об ошибке в ее работе. Типы функций и значения ошибок взаимоувязаны.

|  |  |
| --- | --- |
| **Тип ф-и** | **Значение ошибки** |
| VOID | Такие ф-и почти всегда выполняются успешно. Таких ф-й достаточно мало. Здесь понятие ошибки специфично |
| BOOL | Если вызов заканчивается неудачно, возвращается ноль(не логическое). В остальных случаях – значение, отличное от нуля |
| HANDLE | Если ф-я сработала ошибочно, то обычно возвращается NULL. В остальных случаях – идентифицируется объект, которым можно манипулировать. Некоторые HANDLE-ф-и могут возвращать значение INVALID\_HANDLE\_VALUE, равно -1. Чтобы выяснить, сработала ф-я ошибочно или нет, надо смотреть документацию |
| PVOID | Такие ф-и работают с памятью напрямую. Об ошибке говорит значение NULL. В остальных случаях – возвращается адрес блока данных в памяти |
| LONG/DWORD | Данные ф-и являются счетчиками или указателями. Если ф-я не считала и сосчитать значение счетчика, она возвращается 0 или -1 |

При возникновении ошибки в Win-приложениях надо разобраться, почему вызов ф-и оказался неудачным. Для анализа каждая ошибка кодируется 32-битным числом, которое записывается в локальную память потока, в котором произошла ошибка. Этот механизм позволяет отслеживать ошибки для каждого потока отдельно => ошибка одного потока не может изменить код ошибки другого потока. Когда ф-я, выполнившаяся с ошибкой, вернет управление ОС, ее значение говорит об ошибке и в каких условиях эта ошибка произошла. Получить код ошибки можно с помощью ф-и GetLastError(), которая выдает код последней ошибки в потоке. Данная ф-я возвращает код последней ошибки в потоке, т.к. локальная память каждого потока может хранить только один код ошибки. Чтобы отловить каждую ошибку в потоке, данную ф-ю надо вызывать сразу после той, которая совершила ошибку. Все определенные ошибки хранятся в одном файле – winError.h по структуре:

Идентификатор ошибки

Код ошибки

Описание ошибки

Описание ошибки на национальном языке

Чтобы по коду ошибки получить ее описание, юзается ф-я FormatMessage().

32-битный код ошибки разбит на регионы, каждый из которых содержит ин-ю о параметрах ошибки.

31 – 30 биты: код тяжести ошибки. Если 0 – успех, 1 – ин-я, 2 - предупреждение, 3 – ошибка.

29 бит – значение показывает, кем определен код ошибки – Microsoft или юзер. Если 0 – ошибка Microsoft, 1 – ошибка юзера

28 бит – зарезервирован всегда, 0.

27 – 16 биты – значение кода подсистемы, определяемой в Windows. Он связан не только с типом ошибки, но и с регионом адресного пространства, в котором она произошла

15 – 0 биты – код исключения ошибки, определяется юзером или Microsoft

SetLastError() – установка кода ошибки.

**Процессы в Windows**

**Адресное пространство процесса**

Процесс – экземпляр выполняемой программы, который состоит из 2-х компонентов. 1 – объект ядра процесса, через который ОС управляет данным процессом. В ядре хранится статическая ин-я о процессе. 2 – адресное пространство. В нем – код и данные всех exe- и dll-модулей исполняемой программы. Именно здесь находятся области памяти, динамически распределяемые для стеков потоков, для самих потоков и других нужд. Все процессы инертны. Чтобы процесс что-то выполнил, в нем надо создать минимум один поток. Именно потоки отвечают за исполнение кода в адресном пространстве. Каждый процесс может владеть несколькими потоками, которые одновременно выполняются в адресном пространстве процесса.

Чтобы все потоки в процессе работали, ОС отводит каждому из них определенное процессорное время. Данные отрезки времени – кванты и распределяются между потоками по принципу карусели. При создании процесса первый поток в системе создается автоматически ОС. Если поток не создается, то процесс сразу умирает. Следующие потки могут создаваться как ОС, так и юзером, но они всегда являются дочерними по отношению к главному.

**Типы приложений Windows**

**Процедура сборки и компиляции**

В зависимости от процедуры сборки win-приложения есть 2 типа приложений, в которых процедура сборки и компоновки кардинально отличаются.

1. Приложения с графическом интерфейсом – GUI-приложения
2. Консольные приложения – CUI.

В последних версиях Windows визуальные данные приложения сложно отличить, т.к. истинная консоль, как правило, закрыта от юзера. Максимум – эмуляция консольной строки. Собственно консольные приложения выполняются в окне. У GUI-приложений весь интерфейс чисто графический. Они создают окна, имеют меню, способны организовывать диалог через диалоговые окна и взаимодействовать с юзером через windows-сообщения.

CUI-приложения всегда работают в текстовом режиме. Даже если текстовый режим представлен в главном окне. Приложения не формируют окон, не способны реагировать на действия юзера, не обрабатывают сообщений и не требуют графический библиотек Windows.

При создании проекта приложения Windows компоновщику нужно указать ключ подсистемы, на основе которой будет собираться Win-приложение. Если компоновщик получает ключ /SUBSYSTEM:CONSOLE, он понимает, что нужно произвести компоновку консольного приложения с подключением всех необходимых библиотек. Для граф. приложений и их компоновки ключ - /SUBSYSTEM:WINDOW.

Когда юзер запускает приложение, загрузчик ОС проверяет номер/тип подсистемы приложения, хранящиеся в заголовке exe-файла, и определяет, что за приложение нужно запустить: графические или консольное. Если тип указывает на консольное приложение, то загрузчик создает текстовое консольное окно и «привязывает» к нему исполняемое консольное приложение; иначе – программа просто загружается в память.

В любом Win-приложении должна быть входная ф-я, за реализацию которой отвечает разработчик. В Windows для загрузчика ОС есть 4 типа этой ф-и:

1. WinMain
2. wWinMain
3. main
4. wmain

Данная ф-я фактически не является входной для ОС. Вместо этого происходит обращение к стартовой ф-и из библиотек Си. Эта ф-я инициализирует все нужные библиотеки (именно поэтому в win- приложениях не надо подключать библиотеки), инициализирует ф-и по работе с памятью, обеспечивает корректное создание любых объявленных глобальных и статических Си-объектов. Только после этого вызывается входная ф-и Win-приложения и передается ей управление.

|  |  |  |
| --- | --- | --- |
| **Тип приложения** | **Входная ф-и** | **Стартовая ф-и Windows** |
| GUI, работающая с ANSI | WinMain | WinMainCRTStartUp() |
| GUI, работающая с UNICODE | wWinMain | wWinMainCRTStartUp() |
| CUI, работающая с ANSI | main | mainCRTStartUp() |
| CUI, работающая с UNICODE | wmain | wmainCRTStartUp() |

Нужную стартовую ф-ю в библиотеке Си выбирает компоновщик при сборке exe-файла. Если он находит ключ «…WINDOW», то в коде он ищет 2 ф-и WinMain и wWinMain. Если нет ни одной, то ошибка «Неразрешенный внешний символ» и аварийно завершает сборку приложения. Иначе – вызывается соответствующая стартовая ф-я и ее код встраивается в exe-файл.

Большинство IDE для Win-приложений имеют одно грубое допущение: они не предлагают для компоновщика выбора в типе приложения (по дефолту всегда ключ «…WINDOW»), а для данного ключа подключает все библиотеки Windows, которые могут понадобиться оконному приложению, создает по дефолту главное окно приложения и скрывает его от юзера. Практически все IDE не разрешают изменить данный порядок компоновки юзеру, а если и разрешают (не запрещают изменять эти параметры), то не предоставляют для этого никаких средств.

**Автоматически подключаемые библиотеки**

1. KERNEL32.DLL

Ядро ОС, память, загрузчик ОС

1. USER32.DLL

Диалоги, регистрация событий, работа юзера

1. GDI32.DLL

Графика

1. ADWAPI32.DLL

Работа с железом – драйверы

*Если разрядность выше, чем 32, то…*

**Архитектура памяти Windows**

Для выполнения любой программы в Windows нужна память, в которой она будет хранится как данные, которыми она манипулирует, так и код, который выполняется. Эта память по отношению к работающему процессу является виртуальной и по определению самой Windows – эта область памяти закрыта => если в процессе выполняется какой-либо поток, то он может получить доступ только к той памяти, которой владеет ее процесс. Если в процессе есть несколько поток, они все имеют доступ к одной и той же области памяти, которая закреплена за их процессом. Важно, что адресное пространство является виртуальным => для того, чтобы процесс мог обратиться по какому-либо адресу в этом виртуальном пространстве, его надо спроецировать на физическую память.

Каждое виртуальное адресное пространство разбивается на разделы и их назначение и размер зависят от ядра и версии Windows.

Разделы виртуального адресного пространства:

1. *Раздел для выявления нулевых указателей.*

Резервируется для того, чтобы разработчики программ могли выявлять ошибки в указателях программ и определять ссылку на несуществующие адреса памяти. Любая попытка чтения или записи по адресам данного региона вызывает нарушение доступа. Если ф-я выдает подобную ошибку, то ее возвращаемое значение всегда NULL. При данном значении ф-я, вызывающая ошибку, пытается обратиться повторно по адресу 0000 (8 или 16 нулей).

1. *Для совместимости с различными версиями ОС и DOS-программами.*

Этот раздел появился и развивался с каждой новой версией Windows и нужен, чтобы обеспечить запуск Win-приложений, созданные под платформы другой разрядности. В данном разделе находится код программ, требующих совместимости, но этот раздел закрыт для запускаемого приложения и доступ к нему имеет только ОС. При запуске программ разной разрядности, у каждой есть свой регион в этом разделе, он обособлен, ОС следит, чтобы из данных регионов не было ссылок на другие регионы адресного пространства.

1. *Для кода и данным пользовательского режима.*

Это раздел представляет собой закрытую, неразделяемую часть адресного пространства процесса. Ни один процесс не может получить доступ к данных другого процесса, размещенного в данном разделе (здесь приложение имеет полный доступ только к своему разделу). Основной объем данных, принадлежащих процессу (данные процесса, кода процесса, ссылки на библиотеки и сторонние ресурсы, статические и глобальные переменные и объекты. Здесь нет никаких ограничений у приложения. Сюда, после запуска процесса, подгружается доп. информация, не имеющая отношения к процессу, но нужная для выполнения: часть кода ядра, драйверы устройств, кэш-буферы IO, области памяти, не сбрасываемые в файл подкачки, таблицы для контроля страниц памяти в процессе, информация из файловой системы, если она нужна процессу).

1. *Закрытый раздел.*

Этот раздел заблокирован Microsoft и любая попытка обращения к нему приводит к нарушению доступа. Этот раздел резервируется, чтобы упростить внутреннюю реализацию ОС и нужен для организации взаимодействия приложения и ОС. Информация о том, где хранить очередь сообщений и какие сообщения относятся к данному приложению. Информация из очереди сообщений, относящихся к данному приложению. Этот раздел заблокирован и приложение прямого доступа к нему не имеет, но обратиться к нему может ОС (или ее отдельный процесс, запущенный в режиме ядра).

1. *Для общих файлов, проецируемых в память.*

В этом разделе хранятся код и данные, разделяемые между всеми запущенными процессами (независимо от разрядности и раздела, в котором хранятся данные процесса). Сюда грузятся самые первые 4 библиотеки. Данные в этом разделе приложениями только читаются, практически изменить они не могут, за редким исключением, если оно имеет приоритет 0. Изменять такие приложения могут только данные в своих подгружаемых библиотек. Сюда грузятся все библиотеки, на которые ссылается приложение, здесь же хранятся объекты, связанные с этими библиотеки и могут появляться сообщения, генерируемые кодом библиотек.

1. *Для кода и данных режима ядра.*

Этот раздел очень специфичен и юзается для сетевой поддержки Windows. Сюда помещаются драйверы устройств и код низко уровневого управления сетевыми потоками, код управления памятью и файловой системой, код сетевых протоколов, сетевых драйверов и подключений. Эти данные защищены, доступ из них можно получить только по ряду ограниченных адресов (адреса, доступ к которым открыт данной ОС). Поток, который пытается обратиться по одному из адресов памяти в данном разделе, вызовет нарушение доступа в случае, если поток не имеет прав на чтение этих данных.

Какая бы программа ни запускалась и какой бы разрядности ни была платформа данные разделы всегда есть в адресном пространстве и их использование определяет ОС.

**Библиотеки Windows**

Динамически Подключаемые библиотеки – снова функционирования ОС, т.к. весь код ядра, системных компонент и утилит реализованы в виде библиотек. С развитием ОС в библиотеки стали помещать не только исполняемый код, но и отдельные компоненты, которые требуются приложениям только по мере их требования. Это позволило сократить код и объем исполняемых компонент => добиться экономии виртуальной памяти. При юзании библиотек удалось добиться следующих улучшений:

1. Расширение функциональности приложений. DLL можно грузить в адресное пространство процесса динамически => приложение, определив, какие действия от него требуется, подгружает только нужный код.
2. Возможность юзания различных языков программирования. Т.к. DLL – откомпилированный компонент, приложение и ОС все равно, на каком языке реализован первоначальный код. Это же предоставлено и разработчикам, сняв с них ограничение в выборе языка реализации.
3. Более простое управление проектом. Если в процессе разработки ПО отдельные части создаются разными группами, то при юзании библиотек упрощается взаимодействие этих групп. Достаточно каждой библиотеке сделать описание реализованных ф-й и предоставить их другим участникам проекта.
4. Экономия памяти. Загрузчик ОС при необходимости может подгружать в адресное пространство не только код библиотеки, но и код отдельных ф-й, если загрузка динамическая.
5. Экономия и разделение ресурсов. В библиотеке можно реализовать такие ресурсы, как: шаблоны диалоговых окон, битовые карты, растровые изображения, значки, иконы и отдельны классы окон, элементы локализованного интерфейса.
6. Упрощение локализации ОС и приложений.
7. Решение проблем, связанных с особенностями платформ (с появлением 8-ки и 10-ки – почти самое важное улучшение). К библиотеке одной разрядности можно подключить библиотеку другой разрядности.

**Лекция 5**

**Этапы создания библиотек.**

Для создания библиотеки в зависимости от ее наполнения и ее экспорта/импорта, можно выделить следующие этапы создание динамически подключаемой библиотеки:

1)Прежде всего необходимо подготовить заголовочный файл с прототипами функций, структурами и идентификаторами, экспортируемыми из библиотеки. Данный файл включается в исходный код всех модулей будущей библиотеки

2)Ос-ся реализация исходного кода в самой библиотеке с полным определением всех функций и переменных, который должна использовать библиотека

3)В идеале, если исходный код библиотеки один, то на него создается один объектный файл, который и используется при сборке исполняемого модуля. Если файлов библиотеки несколько, то создается несколько объектных файлов. По одному на каждый модуль.

4)Компоновщик собирает все объектные модули, если их несколько, в один загрузочный dll-модуль, в который в конечном итоге помещаются двоичный код и переменные(глобальные и статические, которые использует данная библиотека или их экспортирует).Данный файл необходим при компиляции исполняемого модуля библиотеки

5) Если компоновщик обнаружит, что библиотека экспортирует хотя бы одну переменную или функцию, то она создаст еще и LIB-файл для данной библиотеки. Данный файл по размеру совсем крошечный, тк содержит только список символьных имен функций и переменных, экспортируемых из библиотеки. Итоговая компиляция библиотеки также невозможна без файла LIB

6)Загрузочный модуль библиотеки( или несколько модулей) и LIB-файл(при его наличии) компилируются в результате чего получается исполняемый модуль библиотеки, который можно использовать и подключать к любому приложению.

**Дома самостоятельно посмотреть код, который будем на лабе делать. Просто найти, как пишется исходный код, как создается объектный файл, как пишется заголовочный, как создается** LIB **файл и как происходит итоговая компиляция.**

Компилируя любую библиотеку следует помнить, что она представляет собой всего лишь набор автономных функций, который могут использоваться любой программой Windоws, несмотря на то, на каком ЯП они реализованы. Следует помнить, что в библиотеке нет кода, предназначенного для обработки циклов сообщений, для создания окон и управления ими, при компоновке библиотеки следует учитывать, что процедура ничем не отличается от компоновщик исполняемого модуля(exe). За исключением того, что компоновщику необходимо указать ключ /DLL. В этом случае, компоновщик записывает в исполняемый файл библиотеки(в его заголовочную часть) информацию, по которой загрузчик ОС понимает, что это библиотека, а не exe- модуль. При этом, действия загрузчика также отличаются от его действий при загрузке исполняемого модуля. Загрузчик определяет, какое приложение вызывает библиотеку, находит адресное пространство данного процесса, а в нем необходимый раздел для загрузки библиотеки и проецирует тело библиотеки в найденное адресное пространство. При этом, любая библиотека теряет свою индивидуальность. Для потоков и процесса код и данные библиотеки лишь дополнительный код, оказавшийся в адресном пространстве. Кроме того, любые созданные библиотекой объекты( точнее, функции этой библиотеки) принадлежат вызывающему процессу. DLL в принципе, ничем владеть не может.

При загрузке библиотеки в адресное пространство вызывается функция выделения памяти, которая резервирует регион в адресном пространстве, куда будет загружена библиотека. Если библиотека будет «выгружена» из адресного пространства процесса(процедура выгрузки для библиотеки невозможна) зарезервированный за библиотекой регион не освободится, тк система не фиксирует того, что данный регион был зареган для этой либы.Освобождение данного региона возможно только в том случае, когда вызывается функция освобождения памяти и процесс завершается, те адресное пространство разрушается. На основании этого, выделяют два вида связывания либы и связываемого модуля(вызывающего процесса):

1)Явное связывание библиотеки

Возможно только при использовании функции LoadLibrary() или ее разновидностей. В этом случае, ссылки на библиотеку устанавливаются в момент вызова функции и в адресное пространство процесса попадает весь код указанной библиотеки.

2)Неявное связывание библиотеки

Указывается ссылка на библиотеку в разделе описания внешних ссылок. После указания ссылки на библиотеку в коде программы при вызове любой из функций библиотеки, данная функция загружается в адресное пространство вызывающего процесса и исп-ся в исполняемом файле. Следует учесть, что при неявном связывании в адресное пространство попадает не весь код библиотеки, а только то, что было вызвано в ходе работы процесса. Несмотря на то, каким способом связывалась библиотека, процедура выгрузки для библиотеки идентична в обоих случаях. Производится она с помощью функции FreeLibrary(), которая уничтожает ссылки на библиотеку, оставляя код библиотеки в адресном пространстве. **Самостоятельно можно посмотреть функции по работе с памятью в Windows,(деление региона, запись в память)**