Динамически загружаемые библиотеки. Использование и механизмы загрузки.

Библиотека — сборник подпрограмм или объектов, используемых для разработки программного обеспечения (ПО). Библиотека —это «сборник» кода, который можно многократно использовать в самых разных программах.   
  
Библиотека DLL может определять два типа функций: экспортированные и внутренние. Экспортированные функции должны вызываться другими модулями, а также из библиотеки DLL, в которой они определены. Внутренние функции обычно предназначены для вызова только из библиотеки DLL, в которой они определены. Хотя библиотека DLL может экспортировать данные, ее данные обычно используются только его функциями. Однако нет ничего, чтобы предотвратить чтение или запись этого адреса другим модулем.

Библиотеки DLL позволяют разбивать приложения на модули, чтобы их функциональные возможности можно было обновлять и использовать повторно. Библиотеки DLL также помогают сократить затраты на память, если несколько приложений используют одни и те же функции одновременно  
  
  
Библиотеки можно разделить на два типа, статические и динамические.

* Статические – состоит из подпрограмм, которые компилируются и линкуются непосредственно с Вашей программой. При компиляции программы использует статическую библиотеку, весь функционал статической библиотеки становится частью исполняемого файла.
* Динамические - состоят из подпрограмм, которые подгружаются в вашу программу во время её выполнения. При компиляции программы, которая использует динамическую библиотеку, эта библиотека не становится частью вашего исполняемого файла —она ​​так и остается отдельным модулем. В windows имеют формат .dll (Dynamic link library), на unix-подобных системах аналоги это так называемые разделяемые объекты (shared objects .so), на mac os - .dylib

Преимущества динамической библиотеки:

* Динамическая компоновка экономит место на диске и пропускную способность. Это связано с тем, что несколько приложений могут одновременно использовать одну копию библиотеки DLL. В отличие от этого, каждое приложение, использующее статическую библиотеку, имеет код библиотеки, связанный с исполняемым образом. Тем самым, увеличивается занимаемое на диске место и используемая для передачи данных пропускная способность.
* Простота обслуживания, обновления или поддержки приложения. Если возникла необходимость изменить библиотеку DLL, то можно обновить только её, при этом используемые ею приложения перекомпилировать не нужно.
* С помощью явной компоновки можно обнаруживать и загружать библиотеки DLL во время выполнения. Например, это могут быть расширения приложения, которые добавляют новые функциональные возможности без повторной сборки и развертывания приложения. Подробнее о способах компоновки мы поговорим позже.
* Динамическая компоновка упрощает поддержку приложений, написанных на разных языках программирования. Программы, написанные на разных языках программирования, могут вызывать одну и ту же функцию в библиотеке DLL при условии соблюдения соглашения о ее вызове. Программы и функция в библиотеке DLL должны отвечать следующим требованиям к совместимости: ожидаемый функцией порядок передачи аргументов в стек; выполнение очистки стека функцией или приложением; передача аргументов в регистрах.

Недостатки динамической библиотеки:

* Программы, использующие динамические библиотеки, должны явно подключать и взаимодействовать с динамической библиотекой. Этот механизм может затруднить взаимодействие с динамической библиотекой. Для упрощения этого процесса используют библиотеки импорта.

**Библиотека импорта**

Библиотека импорта — это библиотека, которая автоматизирует процесс подключения и использования динамической библиотеки. В Windows это обычно делается через небольшую статическую библиотеку (.lib) с тем же именем, что и у динамической библиотеки (.dll). Статическая библиотека линкуется с вашей программой во время компиляции, и тогда функционал динамической библиотеки может эффективно использоваться в вашей программе, как если бы это была обычная статическая библиотека. В Linux общий объектный файл (с расширением .so) дублируется сразу как динамическая библиотека и библиотека импорта. Большинство линкеров при создании динамической библиотеки автоматически создают к ней библиотеку импорта.

**Стандартный порядок поиска классических приложений**

Стандартный порядок поиска БИБЛИОТЕК DLL, используемый системой, зависит от того, включен или отключен безопасный режим поиска DLL.   
Если безопасный режим включен:

* Каталог, из которого загружено приложение.
* Системный каталог. GetSystemDirectory
* 16-разрядный системный каталог.
* Каталог Windows. Используйте GetWindowsDirectory , чтобы получить путь к этому каталогу
* Текущий каталог
* Каталоги, перечисленные в переменной среды PATH. Обратите внимание, что это не включает путь для каждого приложения, указанный в разделе реестра путей приложений. Ключ пути к приложению не используется при вычислении пути поиска библиотеки DLL.

Если безопасный режим выключен:

* Каталог, из которого загружено приложение.
* Текущий каталог.
* Системный каталог. GetSystemDirectory
* 16-разрядный системный каталог.
* Каталог Windows. GetWindowsDirectory.
* Каталоги, перечисленные в переменной среды PATH.

**Альтернативный порядок поиска**

Стандартный порядок поиска, используемый системой, можно изменить, вызвав функцию LoadLibraryEx с LOAD\_WITH\_ALTERED\_SEARCH\_PATH и параметром lpFileName. Стандартный порядок поиска также можно изменить, вызвав функцию SetDllDirectory. Порядок поиска всё также зависит от режима безопасного поиска

* Каталог lpFileName
* Текущий каталог.
* Системный каталог. GetSystemDirectory
* 16-разрядный системный каталог.
* Каталог Windows. GetWindowsDirectory.
* Каталоги, перечисленные в переменной среды PATH.

**Критерии поиска динамической библиотеки**

* Если библиотека DLL с тем же именем модуля уже загружена в память, система проверяет только перенаправление и манифест перед разрешением загруженной библиотеки DLL независимо от того, в каком каталоге он находится. Система не выполняет поиск библиотеки DLL.
* Если библиотека DLL с тем же именем модуля уже загружена в память, система проверяет только перенаправление и манифест перед разрешением загруженной библиотеки DLL независимо от того, в каком каталоге он находится. Система не выполняет поиск библиотеки DLL.
* Если библиотека DLL с тем же именем модуля уже загружена в память, система проверяет только перенаправление (dll redirection) и манифест сборки перед разрешением загруженной библиотеки DLL независимо от того, в каком каталоге он находится. Система не выполняет поиск библиотеки DLL.

**Связывание**

Для связывания с одной и той же библиотекой DLL исполняемый файл может использовать любой из этих способов. Кроме того, они не являются взаимоисключающими, то есть два разных исполняемых файла могут связываться с одной библиотекой DLL разными способами. Решение о применении неявного или явного связывания принимается на уровне архитектуры приложения. Каждый из этих способов имеет свои преимущества и недостатки.

Исполняемый файл можно связать с библиотекой DLL (загрузить ее) одним из двух способов:

* Неявное связывание — операционная система загружает библиотеку DLL в тот момент, когда она используется исполняемым файлом. Исполняемый файл клиента вызывает экспортированные функции библиотеки DLL так же, как статически скомпонованные и включенные в состав самого исполняемого файла функции. Процесс неявного связывания также иногда называют статической загрузкой или динамической компоновкой времени загрузки.
* Динамическая компоновка упрощает поддержку приложений, написанных на разных языках программирования. Программы, написанные на разных языках программирования, могут вызывать одну и ту же функцию в библиотеке DLL при условии соблюдения соглашения о ее вызове. Программы и функция в библиотеке DLL должны отвечать следующим требованиям к совместимости: ожидаемый функцией порядок передачи аргументов в стек; выполнение очистки стека функцией или приложением; передача аргументов в регистрах.

**Неявное связывание**

Неявное связывание происходит в момент, когда из кода приложения вызывается экспортированная функция библиотеки DLL. При компиляции или сборке исходного кода вызывающего исполняемого файла для вызова функции DLL в коде объекта создается ссылка на внешнюю функцию. Для разрешения этой внешней ссылки приложение должно связаться с библиотекой импорта (LIB-файл), которая предоставляется разработчиком библиотеки DLL.

Библиотека импорта содержит только код для загрузки библиотеки DLL и реализации вызовов ее функций. При поиске внешней функции в библиотеке импорта компоновщик определяет, что код этой функции находится в библиотеке DLL. Для разрешения внешних ссылок на библиотеки DLL компоновщик просто добавляет в исполняемый файл сведения, на основании которых система определяет, где следует искать код DLL при запуске процесса.

При запуске программы, которая содержит динамически связываемые ссылки, система использует сведения из исполняемого файла для поиска необходимых библиотек DLL. Если найти библиотеку DLL не удается, система завершает процесс и отображает диалоговое окно с сообщением об ошибке. В противном случае система сопоставляет модули DLL в адресном пространстве процесса.

Если любая из библиотек DLL содержит функцию точки входа для инициализации и завершения кода, например DllMain, операционная система вызывает эту функцию. С помощью одного из параметров, передаваемого в функцию точки входа, задается код, который указывает на связывание библиотеки DLL с процессом. Если функция точки входа не возвращает значение TRUE, система завершает процесс и возвращает ошибку.

Наконец, система изменяет код исполняемого файла процесса и предоставляет функциям DLL начальные адреса.

Как и в случае с остальном кодом программы, загрузчик сопоставляет код DLL в адресном пространстве процесса при запуске процесса. Операционная система загружает его в память только при необходимости. В связи с этим атрибуты кода PRELOAD и LOADONCALL, которые использовались DEF-файлами для управления загрузкой в предыдущих версиях Windows, более не имеют смысла.

**Явное связывание**

В большинстве приложений используется более простой способ неявного связывания. Тем не менее иногда явное связывание необходимо. Такой подход может требоваться в следующих случаях:

* Имя библиотеки DLL, которую необходимо загружать, становится известно приложению только во время выполнения. Например, приложение может получать имя библиотеки DLL и экспортированные функции из файла конфигурации во время запуска.
* Если при запуске не удается найти нужную библиотеку DLL, процесс, в котором используется неявное связывание, завершается операционной системой. Процесс с явным связыванием в таких ситуациях не завершается и может попытаться восстановиться после ошибки. Например, процесс может уведомить пользователя об ошибке и запросить указать другой путь к библиотеке DLL.
* Процесс с неявным связыванием также завершается, если в любой из связанных библиотек DLL функция DllMain завершается сбоем. Процесс с явным связыванием в таких ситуациях не завершается.
* Приложение, в котором применяется неявное связывание с множеством библиотек DLL, может долго запускаться, поскольку операционная система Windows при загрузке приложения загружает все библиотеки DLL. Чтобы ускорить процесс запуска приложения, можно выполнять неявное связывание только с теми библиотеками DLL, которые необходимы непосредственно после загрузки. Другие библиотеки DLL могут загружаться позднее по мере необходимости посредством явного связывания.
* При явном связывании приложению не требуется библиотека импорта. Если из-за изменений в библиотеке DLL изменяются порядковые номера экспорта, приложениям не нужно будет повторно выполнять связывание, если для вызова GetProcAddress в них используется имя функции, а не ее порядковый номер. Если приложение использует неявное связывание, в случае изменений в библиотеке импорта им потребуется выполнить связывание повторно. (GetProcAddress – функция, возвращающая адрес экспортируемой функции или переменной)

**DllMain**

Необязательная точка входа в библиотеку динамической компоновки (DLL). При запуске или завершении процесса или потока система вызывает функцию точки входа для каждой загруженной библиотеки DLL с помощью первого потока процесса. Система также вызывает функцию точки входа для библиотеки DLL при загрузке или выгрузке с помощью функций LoadLibrary.

Библиотека DllMain предназначена для выполнения минимальных задач инициализации с помощью небольшого подмножества API Microsoft Windows. Невозможно вызвать любую функцию в DllMain, которая напрямую или косвенно пытается получить блокировку загрузчика. В противном случае вы введете возможность взаимоблокировки или сбоя приложения. Ошибка в реализации DllMain может поставить под угрозу весь процесс и все его потоки.

Соответственно, функция точки входа должна выполнять только простые задачи инициализации или завершения. Он не должен вызывать функцию LoadLibrary (или функцию, которая вызывает эту функцию), так как это может создавать циклы зависимостей в порядке загрузки DLL. Идеальной библиотекой DllMain будет просто пустая заглушка. Однако, учитывая сложность многих приложений, это, как правило, слишком ограничительно. Хорошее правило для DllMain заключается в том, чтобы отложить как можно больше инициализации. Отложенная инициализация повышает надежность приложения, так как эта инициализация не выполняется во время блокировки загрузчика. Кроме того, отложенная инициализация позволяет безопасно использовать гораздо больше api Windows.