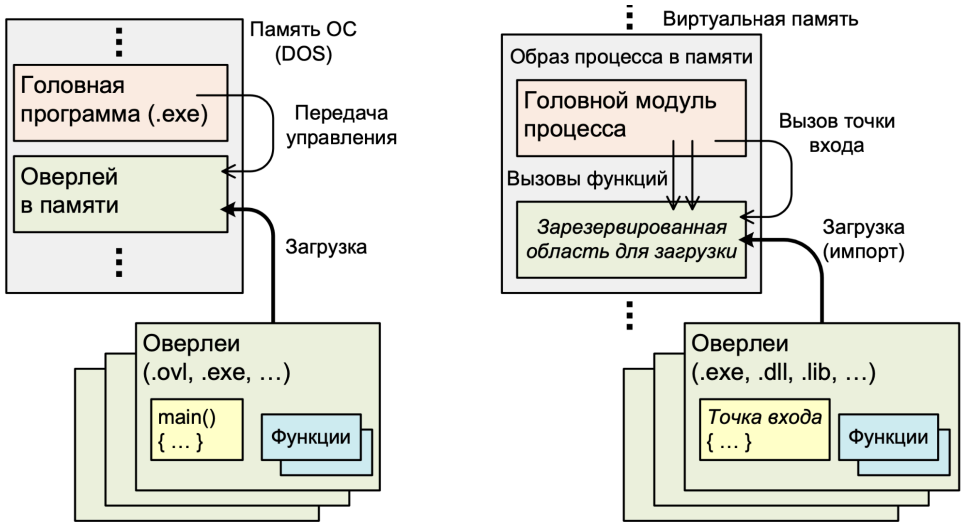
Некоторые служебные и технологические задачи

Оверлеи (overlay) - программные библиотеки, загружаемые и исполняемые в составе других программ, не находятся постоянно в памяти, а сменяют друг друга при необходимости.
Их использование позволяет экономить память, обеспечивать гибкость формирования конфигурации программы при загрузке и структурировать программу из готовых модулей.
Оверлеи реализованы в динамических библиотеках (DLL) в Windows, разделяемых библиотеках в Unix. Эффективное использование требует поддержки со стороны операционной системы, что в DOS могло ограничивать применение оверлеев из-за ограничений в управлении памятью.



Оверлеи

Динамически подключаемые библиотеки (DLL, Dynamic Link Library) в Windows являются исполняемыми модулями, которые не могут выполняться самостоятельно, а присоединяются к другим исполняемым файлам. Они имеют аналогичную структуру с .EXE-файлами в формате Portable Executable (PE). DLL обладают следующими особенностями:

- Они включаются в адресное пространство процесса, обладают собственными экземплярами.
- Загружаются через механизмы создания процесса, счетчик привязок управляет загрузкой/выгрузкой.
- Экспортируют символы (функции, методы) через таблицу экспорта.

DLL служат для повторного использования кода, уменьшения размера исполняемых файлов и упрощения обновлений программного обеспечения. Могут содержать ресурсы помимо функций и данных.

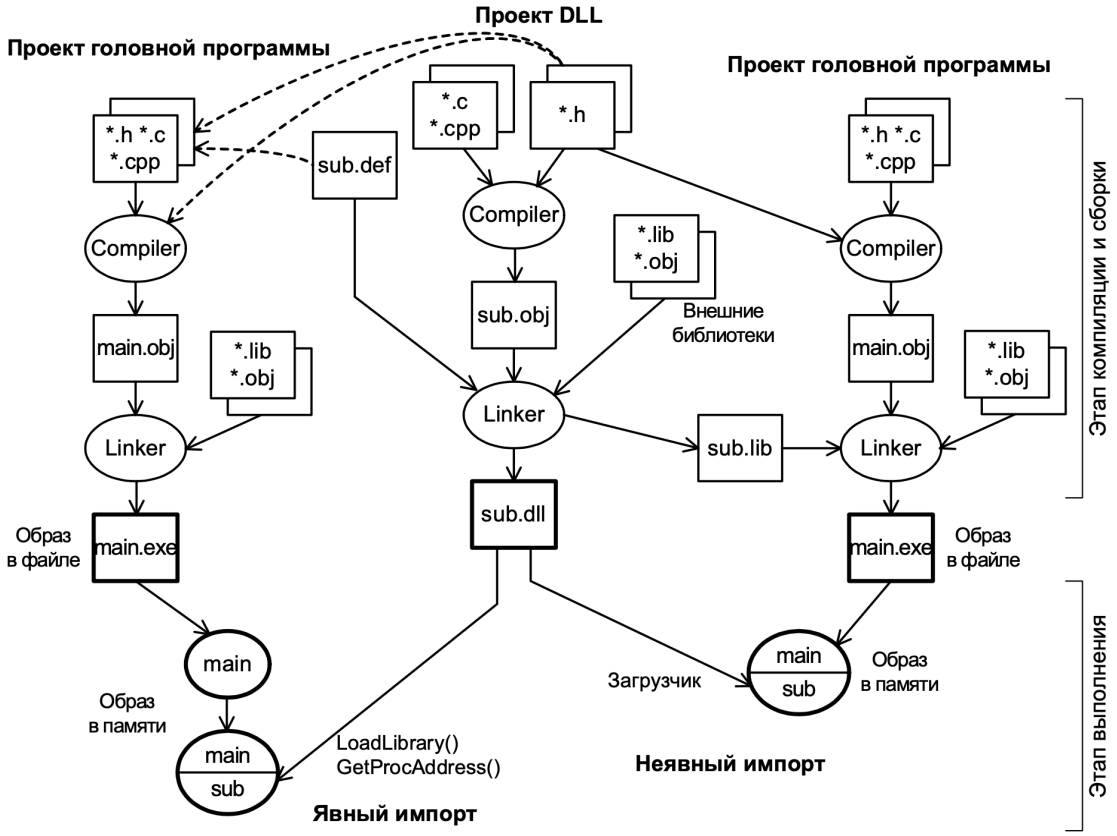
Виды библиотек: Существуют статические и динамические библиотеки. **Статические** библиотеки компилируются непосредственно в исполняемый файл, а **динамические** библиотеки загружаются в память при выполнении программы.

Динамическое подключение DLL происходит во время выполнения программы, в то время как **статическое** подключение осуществляется на этапе компиляции.

Подключение (импорт) DLL:

- Явный импорт DLL - в процессе выполнения программы через системные вызовы, такие как *LoadLibrary, GetModuleHandle, FreeLibrary, u GetProcAddress*. Этот способ обеспечивает гибкость и управляемость, но более трудоемок.
- Неявный импорт DLL - во время линковки программы, функции вызываются так же, как если бы они были частью самой программы, а не внешней библиотекой, в ней импортируемые символы декларируются через заголовочные файлы и библиотеки. Этот способ более прост в использовании, но требует соблюдения единых правил и форматов вызова функций.

При использовании DLL важно согласовывать формат вызова функций между библиотекой и программой, обратная совместимость может быть обеспечена использованием стандартного формата *__stdcall*. Экспорт и импорт классов может быть сложным из-за необходимости согласования их описаний и методов.



Динамически подключаемая библиотека (DLL)

Программирование DLL (Dynamic Link Libraries) в Windows предполагает создание библиотек, которые могут быть загружены и использованы другими программами.

1 Точка входа:
В DLL обязательно должна быть точка входа *DllMain*, которая описывает функцию:
BOOL APIENTRY DllMain(HANDLE hModule, DWORD fCall, LPVOID lpReserved);
Параметр *fCall* указывает на тип обращения к точке входа, например, DLL_PROCESS_ATTACH, DLL_PROCESS_DETACH, и т. д.

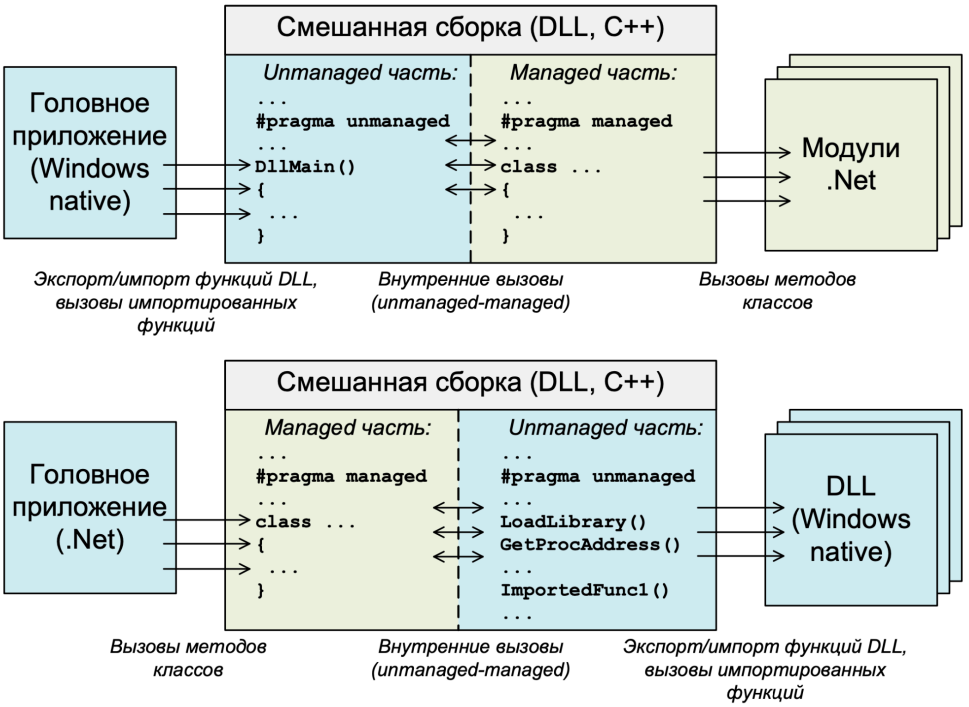
2 Явный импорт:
Для явного импорта DLL требуется знание имен символов для импорта.
Обычно решается предоставлением соответствующего заголовочного файла (.h) и документацией.

3 Неявный импорт:
Необходимо соблюдать правила оформления функций в DLL.
В Visual C++, экспортируемые символы объявляются с *__declspec(dllexport),* а импортируемые - с *__declspec(dllimport).*

4 Проблемы соглашений об именах:
При экспорте DLL по умолчанию используются "внутренние" имена, что может затруднить связывание.
Решение — явное стандартное описание символов для экспорта.

5 Средства описания символов для экспорта:
В C-ориентированных средствах обычно используется .def-файл, где секция EXPORTS описывает экспортируемые имена.

Смешанные сборки (Mixed Assembly) в .NET позволяют объединить в одном проекте модули .NET и "обычные" (Native Win API) для взаимодействия между managed и unmanaged кодом. Это полезно при модификации старых продуктов, требующих дополнения .NET кодом, или при работе с Win API в рамках .NET. В Visual Studio это реализуется через проекты на C++, поддерживающие managed и unmanaged версии.
Маршаллинг параметров и ключевые слова ref и out используются для вызовов между managed и unmanaged кодом. Хотя можно импортировать функции DLL в managed код, использование смешанных сборок предпочтительнее для сложных взаимодействий.



Конфигурирование и настройка

Основной принцип – отделение программ от данных. Существуют два основных подхода: централизованный и децентрализованный.

В **децентрализованном** подходе каждое приложение самостоятельно управляет конфигурацией, обычно храня её в текстовых файлах (.ini, .cfg, XML). Этот подход обеспечивает гибкость, универсальность и независимость, но требует документирования структуры данных.

Централизованный подход предполагает центральное хранилище конфигурационных данных, управляемое специализированной службой с API для доступа. Этот подход обеспечивает унификацию доступа и разгружает приложения от управления конфигурациями, но зависит от качества системной поддержки.

Реестр Windows - это специальная база данных, где приложения и операционная система хранят разнообразную информацию, включая данные об аппаратной конфигурации, установленных программах, драйверах, административных настройках и пользовательских данных. Реестр организован иерархически как дерево из ключей и значений, с ограничением в 512 уровней.

Физический уровень представления реестра – файлы. В Windows 7 это директорий *%windir%\System32\config*
и файлы *DEFAULT, SAM, SECURITY, SOFTWARE, SYSTEM.*
Hive (букв. «улей») – образ данных загруженный в память реестра.

Логический уровень представления – иерархический список (дерево).

Ключи представляют собой каталоги, содержащие подключи и значения, а также могут быть ссылками на другие ключи. Значения в реестре характеризуются типами данных, такими как строки, двоичные данные, двойные слова и другие. Рекомендуется хранить данные конфигурации приложений в определенных разделах реестра, в зависимости от их общности или специфичности для пользователей.

При использовании реестра важно соблюдать правила, такие как запись только данных конфигурации и инициализации приложений, хранение больших данных в файлах, а не в реестре, избегание хранения исполняемого кода в реестре и группировка данных для экономии места. При удалении приложения необходимо удалять из реестра все связанные с ним данные.

В целом, реестр Windows является важным механизмом хранения настроек и данных, но требует внимательного и аккуратного обращения для предотвращения ошибок и обеспечения эффективной работы системы.

Regedit - утилита для работы с реестром Windows. Она позволяет работать с реестром через GUI и выполнить операции по изменению данных с помощью файлов .reg в командном режиме. Файлы .reg содержат данные реестра для импорта и экспорта.
Режимы работы regedit:

- Интерактивный – используя GUI (отображение дерева реестра, поиск и модификация его узлов)
- Командный (пакетный) – выполнение «сценариев» (файлы .reg)

Функции API для работы с реестром Windows позволяют программистам взаимодействовать с реестром операционной системы. Некоторые ключевые функции включают:

- Создание и открытие ключей реестра: RegCreateKey, RegCreateKeyEx, RegOpenKey, RegOpenKeyEx
- Закрытие и удаление ключей: RegCloseKey, RegDeleteKey
- Экспорт и импорт данных реестра: RegSaveKey, RegLoadKey
- Получение списка подключей и значений: RegEnumKey, RegEnumKeyEx, RegEnumValue
- Чтение и запись значений: RegQueryValue, RegQueryValueEx, RegSetValueEx
- Создание оповещений об изменениях значений: RegNotifyChangeKeyValue

Эти функции позволяют программно управлять содержимым реестра, извлекать данные, изменять значения и отслеживать изменения для обеспечения корректной работы программ, использующих реестр Windows.

Протоколирование (журналирование) - фиксация информации о событиях в системе для обнаружения ошибок, отладки, мониторинга процессов, сбора данных и контроля безопасности. Важно для отладки программ, особенно в production-режиме.

Журналирование может быть **децентрализованным** (приложения ведут журналы самостоятельно) или **централизованным** (единая служба для всех приложений). Windows имеет журналы событий (Event Logs) для приложений, системы и безопасности.

Три основных журнала:

- Журнал приложений (Application) – используется прикладными программами
- Системный журнал (System) – используется драйверами и службами
- Журнал системы безопасности (Security)

Событие (Event) – объект или событие, информация о котором заносится в журнал.

События классифицируются по типам: ошибки, предупреждения, информационные сообщения, успешные и неуспешные аудиты. **Журналы хранятся в реестре.**

Для управления журналами в Windows используются системные утилиты и API функции (RegisterEventSource, ReportEvent, OpenEventLog). Программы могут регистрировать собственные события. Файлы журналов представляют собой модули DLL с текстами сообщений, которые компилируются для использования в программах.

Командная строка, или интерфейс командной строки (CLI), представляет собой способ взаимодействия с операционной системой путем ввода текстовых команд. В Windows существует несколько интерпретаторов командной строки, таких как COMMAND.COM, CMD.EXE и PowerShell. PowerShell представляет собой более мощный интерпретатор, который расширяет возможности обычных команд и предоставляет cmdlet'ы для управления системой.

Команды вводятся последовательно, с указанием команды и аргументов. Интерпретатор CLI может работать в интерактивном режиме (ввод команд вручную) или пакетном режиме (выполнение последовательности команд из файла). Расширение файлов для пакетных скриптов в Windows обычно .bat для COMMAND.COM и CMD.EXE, а .ps1 для PowerShell.

PowerShell поддерживает объектно-ориентированную модель, взаимодействие с .NET и CLR (Common Language Runtime), что делает его мощным инструментом для автоматизации задач и управления системой. Он также обладает функциями автодополнения, истории команд, а также возможностью создания и исполнения скриптов. Для безопасности выполнения скриптов в PowerShell необходимо настроить политику выполнения скриптов с помощью команды set-executionpolicy.

Windows Management Instrumentation (WMI) - управление Windows-системой через объекты с методами и свойствами, хранящимися в файлах .mof. Доступ осуществляется через PowerShell.

Проблемы при использовании WMI включают нестабильность объектов, изменения в классах и провайдерах между версиями Windows, а также медленность универсальных приложений для работы с WMI из-за загрузки всей иерархии.

SNMP (Simple Network Management Protocol) и MIB (Management Information Base) представляют протокол и базу информации для управления сетевыми узлами. SNMP позволяет запросы данных, а MIB хранит информацию об узлах в иерархии. Архитектура SNMP включает агентов и менеджеров для управления узлами и изменения их параметров.

WinHook (или Windows Hook) — это механизм в операционной системе Windows, который позволяет перехватывать и обрабатывать события, происходящие в системе или в приложениях. Хуки могут быть глобальными (Разрешают перехватывать сообщения для всех приложений в системе) или локальными (только для конкретного приложения). С помощью хуков разработчики могут получать уведомления о различных событиях, таких как нажатия клавиш, движения мыши и изменения состояния окон.

Что можно перехватывать
С помощью WinHook можно перехватывать следующие события:

- Нажатия клавиш (keyboard hooks)
- Движения мыши (mouse hooks)
- Изменения окон (message hooks)
- Системные события (например, события о входе/выходе пользователя)

Как перехватывается
Перехват событий осуществляется через API-функции Windows. Обычно это делается следующим образом:

1. **Установка хука**: Используя функцию SetWindowsHookEx, разработчик регистрирует функцию обратного вызова (callback), которая будет вызываться при возникновении определенных событий.
2. **Обработка событий**: Когда происходит событие (например, нажатие клавиши), Windows вызывает функцию обратного вызова, где разработчик может выполнить нужные действия.
3. **Снятие хука**: После завершения работы с хуком, его можно снять с помощью функции UnhookWindowsHookEx.

Как это работает на логическом уровне
На логическом уровне процесс работы WinHook можно описать следующим образом:

1. **Регистрация хука**: Приложение вызывает SetWindowsHookEx, указывая тип хука и адрес функции обратного вызова.
2. **Перехват события**: Когда событие происходит, система проверяет зарегистрированные хуки и вызывает соответствующие функции.
3. **Обработка**: Функция обратного вызова обрабатывает событие (например, логирует нажатие клавиши) и может либо пропустить событие дальше в систему, либо предотвратить его дальнейшую обработку.

Что происходит, когда нажимаешь на клавишу

1. **Нажатие клавиши:** При нажатии клавиши на клавиатуре создается событие, которое система определяет как "нажатие клавиши".
2. **Отправка сообщения**: Система генерирует сообщение о нажатии клавиши (например, WM_KEYDOWN) и отправляет его в очередь сообщений.

3. **Перехват хуком**: Если установлен хук на клавиатуру, система вызывает функцию обратного вызова, где можно обработать это событие.
4. **Передача сообщения дальше**: После обработки хук может передать сообщение дальше, позволяя другим приложениям увидеть нажатие клавиши.

Про шифратор и дешифратор
Шифратор и дешифратор — это устройства или алгоритмы, которые преобразуют данные из одного формата в другой. В контексте WinHook и перехвата нажатий клавиш можно говорить о шифровании данных, чтобы скрыть или защитить информацию о нажатиях.

- **Шифратор**: Преобразует нажатия клавиш в зашифрованный формат.
- **Дешифратор**: Преобразует зашифрованные данные обратно в оригинальный формат.

По какой шине будет передаваться нажатие клавиши
Нажатия клавиш передаются по **USB** (если клавиатура подключена через USB) или по **PS/2** (если используется старая клавиатура). Внутри системы сигналы обрабатываются через драйверы, которые превращают физические нажатия в сообщения, которые может обрабатывать операционная система.

Командлеты по сути являются собственными командами PowerShell и **не выносятся в отдельные исполняемые файлы**. Командлеты группируются в модули PowerShell, которые можно загружать по требованию. Командлеты могут быть написаны **на любом компилируемом языке .NET** или на собственном языке скриптов PowerShell.

PowerShell использует для именования командлетов связку *глагол-существительное*. Например, включенный в PowerShell командлет **Get-Command** возвращает полный список всех командлетов, зарегистрированных в командной оболочке. Глагол здесь определяет действие, выполняемое командлетом, а существительное определяет ресурс, в котором он выполняет это действие.

Резервное копирование реестра в Windows:

- Для создания резервной копии реестра в Windows можно использовать встроенную утилиту `regedit`.
- Для этого можно создать резервную копию реестра с помощью одной команды или сценария.

Пример команды PowerShell для создания резервной копии реестра:
```
reg export HKLM\Software\MyApp C:\backup\myapp.reg
```

Восстановление реестра в Windows:

- Для восстановления реестра из резервной копии можно использовать команду `reg import` в PowerShell.

Пример:
```
reg import C:\backup\myapp.reg
```

Работа с реестром через PowerShell:

- PowerShell предоставляет мощные средства для работы с реестром Windows.
- Для чтения, записи и удаления значений реестра можно использовать модуль `PSRegistry`.

Пример чтения значения из реестра:
```
Get-ItemProperty -Path 'HKLM:\Software\MyApp' -Name 'MyValue'
```

Работа с журналами Windows через PowerShell:

- Для работы с журналами Windows в PowerShell можно использовать командлеты `Get-WinEvent`, `New-WinEvent`, `Register-WinEvent` и другие.
- Журналы Windows хранятся в специальных файлах `.evtx`.

Пример чтения событий из журнала:
```
Get-WinEvent -LogName Application -MaxEvents 10
```

Конфигурации в Windows:

- В Windows существует множество конфигураций, таких как системные настройки, настройки приложений, параметры безопасности и другие.
- Отличия между конфигурациями могут быть в области применения, уровне доступа и способе управления.

WinHook:

- WinHook (или Windows Hook) - это механизм в Windows, позволяющий перехватывать и обрабатывать различные события в системе.
- WinHook может использоваться для мониторинга действий пользователя, изменения поведения приложений и других задач.

Библиотеки в программировании:

- Библиотеки - это наборы функций и процедур, которые могут быть использованы различными программами для выполнения определенных задач.
- Различают динамические и статические библиотеки. Динамические подгружаются во время выполнения, статические компилируются в исполняемый файл.
- Оверлеи - это часть исполняемого файла, которая может быть загружена в память по мере необходимости.

Смешанные сборки и .NET:

- Смешанные сборки в .NET - это сборки, содержащие как управляемый, так и неуправляемый код.
- .NET Framework предоставляет среду выполнения для управляемого кода и позволяет взаимодействовать с неуправляемым кодом через механизм P/Invoke.