***Динамически подключаемые библиотеки DLL: понятие, преимущества, механизм работы, явное и неявное связывание.***

***Создайте программу с использование асинхронных методов. Программа должна находить все простые числа из диапазона от А до В, где B<1000000.***

**Динамически подключаемые библиотеки (DLL — Dynamic Link Library)** — это исполняемые файлы, которые содержат код и данные, используемые несколькими приложениями одновременно. В отличие от статических библиотек, которые включаются в исполняемый файл во время компиляции, DLL загружаются в память только во время выполнения программы, когда они действительно необходимы. DLL играют важную роль в модульном программировании и повторном использовании кода.

**Понятие DLL**

DLL представляют собой разделяемые библиотеки, которые предоставляют определенный набор функций, классов или ресурсов, доступных для использования несколькими процессами одновременно. Они позволяют разработчикам разделять функциональность на отдельные модули, которые могут обновляться независимо друг от друга.

**Преимущества DLL:**

Экономия памяти: DLL загружаются в память только один раз, даже если несколько приложений используют их, что позволяет экономить оперативную память.

Модульность: Код разделяется на отдельные модули, что упрощает разработку, тестирование и сопровождение программного обеспечения.

Обновляемость: DLL могут обновляться отдельно от исполняемых файлов, что позволяет исправлять ошибки и добавлять новые функции без необходимости перекомпиляции всех приложений, использующих эту библиотеку.

Повторное использование кода: DLL позволяют повторно использовать один и тот же код в разных приложениях, что снижает объем работы и сокращает время разработки.

Уменьшение размера исполняемых файлов: Исполняемые файлы получаются меньше, поскольку код библиотек не включается непосредственно в них.

Поддержка плагинов: DLL могут использоваться для реализации плагин-архитектуры, позволяя расширять функциональность приложений с помощью внешних модулей.

**Механизм работы DLL:**

Компиляция DLL: Исходный код DLL компилируется в объектные файлы, которые затем связываются в динамическую библиотеку (DLL в Windows, .so в Linux, .dylib в macOS).

Загрузка DLL: Когда приложение, использующее DLL, запускается, операционная система (ОС) загружает DLL в память.

Разрешение зависимостей: ОС находит все необходимые DLL, от которых зависит данная DLL, и загружает их тоже.

Связывание: Приложение связывается с DLL и получает доступ к ее функциям, классам или ресурсам.

Вызов функций: Во время выполнения приложения, код из DLL может быть вызван как если бы он находился непосредственно в приложении.

**Явное и неявное связывание (Dynamic Linking)**

Существует два основных способа связывания приложения с DLL: явное и неявное связывание.

**1. Неявное связывание (Implicit Linking):**

Как работает: При неявном связывании приложение связывается с DLL во время компиляции. Компилятор и линкер используют импортную библиотеку (.lib в Windows, .a в Linux/macOS), которая содержит информацию о функциях, экспортируемых DLL.

**Процесс:**

Создание импортной библиотеки: Во время сборки DLL создается импортная библиотека, содержащая информацию о экспортируемых из DLL функциях (используется для неявного связывания).

Связывание: При компиляции приложения, использующего DLL, линкер использует импортную библиотеку для разрешения вызовов функций DLL.

Загрузка во время выполнения: Операционная система автоматически загружает DLL в память при запуске приложения.

**Преимущества:**

Простота использования: Код приложения выглядит так же, как если бы вызывались обычные функции.

Автоматическая загрузка: DLL загружается автоматически, что не требует дополнительного кода.

**Недостатки:**

Жесткая зависимость: Приложение зависит от конкретной версии DLL.

Загрузка всех DLL: Все DLL, указанные в импортной библиотеке, загружаются в память при запуске приложения (даже если не все функции используются).

Сложности в переносимости: Нельзя просто так заменить dll-ку на другой платформе.

**Пример (Windows):**

При компиляции my\_dll.dll, автоматически создается импортная библиотека my\_dll.lib

При компиляции приложения, использующего my\_dll.dll, my\_dll.lib передается линковщику

При запуске приложения my\_dll.dll автоматически загружается ОС

**2. Явное связывание (Explicit Linking):**

Как работает: При явном связывании приложение самостоятельно загружает DLL в память во время выполнения с помощью специальных API (например, LoadLibrary в Windows, dlopen в Linux/macOS) и затем получает адреса нужных функций из DLL с помощью API (например, GetProcAddress в Windows, dlsym в Linux/macOS).

**Процесс:**

Загрузка DLL: Приложение вызывает API для загрузки DLL (например, LoadLibrary или dlopen).

Получение адреса функции: Приложение вызывает API для получения адреса нужной функции (например, GetProcAddress или dlsym).

Вызов функции: Приложение вызывает функцию через полученный адрес.

Выгрузка DLL: Приложение вызывает API для выгрузки DLL (например, FreeLibrary в Windows, dlclose в Linux/macOS).

**Преимущества:**

Гибкость: Приложение может загружать DLL только тогда, когда это необходимо, и может использовать разные версии DLL.

Поздняя загрузка: DLL загружаются только тогда, когда они действительно используются.

Более управляемая работа: Можно контролировать загрузку, использование и выгрузку dll-библиотеки.

**Недостатки:**

Более сложный код: Требуется дополнительный код для загрузки и выгрузки DLL, а также для получения адресов функций.

Увеличение времени разработки: Необходимость писать дополнительный код для работы с DLL.

**Пример (Windows):**

#include <iostream>

#include <windows.h>

typedef int (\*AddFunc)(int a, int b);

int main() {

HINSTANCE hDll = LoadLibrary(TEXT("my\_dll.dll")); // Загрузка DLL

if (hDll != NULL) {

AddFunc add = (AddFunc)GetProcAddress(hDll, "add"); // Получение адреса функции add

if (add != NULL) {

int result = add(5, 3);

std::cout << "Result: " << result << std::endl;

} else {

std::cerr << "Failed to get function address." << std::endl;

}

FreeLibrary(hDll); // Выгрузка DLL

} else {

std::cerr << "Failed to load DLL." << std::endl;

}

return 0;

}