### 1. Что такое USB, и для чего он используется?

USB (Universal Serial Bus) — стандарт интерфейса для подключения периферийных устройств. Он используется для передачи данных, подключения устройств (клавиатуры, мыши, принтеры, флеш-накопители) и зарядки мобильных устройств.

### 2. Какие типы USB-коннекторов существуют, и какие устройства они обычно поддерживают?

- \*\*USB-A\*\*: классический прямоугольный разъем для компьютеров и периферии.

- \*\*USB-B\*\*: квадратный разъем для принтеров и сканеров.

- \*\*USB-C\*\*: компактный, двусторонний, используется в смартфонах, ноутбуках, планшетах.

- \*\*Micro-USB\*\*: устаревший стандарт для мобильных устройств.

- \*\*Mini-USB\*\*: для старых камер и MP3-плееров.

### 3. Каковы основные различия между USB 2.0, USB 3.0 и USB 3.1?

- \*\*USB 2.0\*\*: скорость до 480 Мбит/с.

- \*\*USB 3.0\*\*: до 5 Гбит/с, поддержка одновременной передачи данных в обе стороны.

- \*\*USB 3.1\*\*: до 10 Гбит/с, улучшенное энергопотребление.

- \*\*USB 3.2\*\* и \*\*USB4\*\*: до 40 Гбит/с, поддержка Thunderbolt.

### 4. Как работает принцип передачи данных через USB?

Данные передаются пакетами. Хост-контроллер управляет подключениями, обмениваясь командами и информацией с устройствами через порты.

### 5. Какие стандарты и спецификации определяют технические характеристики USB?

- \*\*USB-IF\*\* (Форум внедрения USB) — организация, разрабатывающая стандарты.

- \*\*Спецификации USB\*\*: 1.0, 2.0, 3.0, 3.1, 4.0.

- \*\*USB PD\*\*: стандарт для зарядки.

### 6. Какие виды USB-кабелей существуют, и какие они имеют различия?

- \*\*Стандартные\*\*: USB-A, USB-B.

- \*\*Миниатюрные\*\*: Mini-USB, Micro-USB.

- \*\*Современные\*\*: USB-C.

Различаются скоростью передачи, током зарядки и совместимостью.

### 7. Какие преимущества и недостатки USB в сравнении с другими интерфейсами передачи данных?

- \*\*Преимущества\*\*: универсальность, поддержка питания, высокая скорость.

- \*\*Недостатки\*\*: ограниченная длина кабеля, уязвимость к вирусам.

### 8. Какие устройства могут быть заряжены через USB, и как работает стандарт USB Power Delivery (PD)?

Заряжаются смартфоны, ноутбуки, гаджеты. \*\*USB PD\*\* обеспечивает мощность до 240 Вт, автоматически регулирует напряжение и ток.

### 9. Какие меры безопасности могут быть применены для защиты от угроз, связанных с использованием USB-портов?

- Отключение автозапуска.

- Использование защитных адаптеров (USB-condoms).

- Антивирусы и программное ограничение доступа к портам.

### 10. Каковы перспективы развития технологии USB в будущем?

Рост скорости передачи данных, улучшение энергопотребления, внедрение USB4 и совместимости с Thunderbolt.

### 11. Что представляет собой интерфейс Inter-Chip (IC) в контексте USB?

Интерфейс USB-IC используется для связи чипов внутри одного устройства. Минимальное энергопотребление, высокая скорость передачи данных.

### 12. Что такое беспроводной USB?

Беспроводной USB использует \*\*Ultra-Wideband (UWB)\*\* для передачи данных без кабелей на короткие расстояния (до 10 м).

### 13. Что такое метод связи в контексте USB?

- \*\*Control\*\*: управление устройствами.

- \*\*Bulk\*\*: передача больших объемов данных.

- \*\*Isochronous\*\*: потоковые данные с минимальными задержками (аудио/видео).

- \*\*Interrupt\*\*: малые объемы с высоким приоритетом (мыши, клавиатуры).

### 14. Какие особенности физического уровня присутствуют в структуре USB?

Используются дифференциальные сигналы D+ и D-, питание через отдельные линии. Сигналы минимизируют помехи и обеспечивают стабильную связь.

### 15. Какова структура пакетов данных в USB?

Пакеты состоят из:

- \*\*Токена\*\* (адрес устройства).

- \*\*Данных\*\* (информация).

- \*\*Подтверждения\*\* (успех/ошибка).

### 16. Как происходит инициализация USB-устройств?

При подключении устройство отправляет хосту информацию о себе (ID, класс), хост определяет драйвер и устанавливает соединение.

### 17. Как регулируется электропитание USB-устройств?

5 В через стандартный порт, до 240 Вт через \*\*USB PD\*\*. Возможна приоритизация питания в зависимости от устройств.

### 18. Что такое USB OTG?

\*\*On-The-Go\*\* позволяет смартфону/планшету быть хостом для подключения флешек, мышей и других USB-устройств.

### 19. Что такое USB-хаб?

Устройство, позволяющее подключить несколько USB через один порт. Может быть пассивным (без питания) или активным (с внешним питанием).

### 20. Как можно увеличить количество USB-портов на ноутбуке?

Использовать USB-хабы или док-станции с поддержкой дополнительных интерфейсов.

### 21. Какие вызовы и проблемы возникают при работе с USB в области кибербезопасности?

- Распространение вирусов через флешки.

- Подделка устройств (например, USB Killers).

- Угрозы утечки данных.

1. Начало программы:

- Программа начинается с подключения необходимых заголовочных файлов, которые предоставляют доступ к функциям Windows API, работе с USB-устройствами, и стандартным библиотекам C++.

2. Объявление глобальных переменных и структур:

- Объявляется глобальная переменная `lastSafelyRemovedDeviceId` для хранения идентификатора последнего безопасно извлеченного устройства.

- Определяется структура `Device`, которая содержит информацию об USB-устройстве: имя, аппаратный идентификатор, флаг возможности извлечения и дескриптор устройства.

- Объявляется глобальный вектор `devices` для хранения списка обнаруженных устройств.

3. Функция `updateDeviceList()`:

- Эта функция отвечает за обновление списка USB-устройств.

- Сначала очищается текущий список устройств.

- Затем используется функция `SetupDiGetClassDevsW` для получения информации о классе USB-устройств.

- В цикле перебираются все найденные устройства, и для каждого устройства:

- Получается имя устройства и его аппаратный идентификатор.

- Определяется, является ли устройство извлекаемым.

- Создается объект `Device` и добавляется в вектор `devices`.

- После обработки всех устройств освобождаются ресурсы, связанные с информацией об устройствах.

4. Функция `printDevices()`:

- Эта функция отвечает за вывод списка устройств на экран.

- Сначала очищается консоль.

- Затем выводится заголовок "Список USB-устройств".

- В цикле перебираются все устройства из вектора `devices`, и для каждого устройства выводится его номер и информация (аппаратный ID и имя).

- В конце выводится инструкция для пользователя.

5. Функция `checkForUnsafeRemoval()`:

- Эта функция проверяет, не было ли устройство извлечено небезопасно.

- Используются статические векторы `previousDevices` и `reportedDevices` для отслеживания изменений в списке устройств.

- Сравнивается текущий список устройств с предыдущим.

- Если устройство исчезло из списка, не было безопасно извлечено и о нем еще не сообщалось, выводится предупреждение.

- Обновляется список предыдущих устройств и очищается список сообщенных устройств, которые снова появились.

6. Функция `main()`:

- Устанавливается локаль для корректного отображения русских символов.

- Запускается основной цикл программы:

- Каждую секунду обновляется список устройств, выводится на экран и проверяется на небезопасное извлечение.

- Проверяется ввод пользователя:

- Если нажата 'q' или 'Q', программа завершается.

- Если нажата цифра от 1 до 9, происходит попытка извлечения соответствующего устройства.

- Если устройство успешно извлечено, его ID сохраняется как последнее безопасно извлеченное.

- После каждой операции извлечения список устройств обновляется и выводится заново.

- Цикл повторяется каждые 100 миллисекунд для снижения нагрузки на процессор.

7. Завершение программы:

- После выхода из основного цикла программа завершается, возвращая 0.

#include <windows.h> // Подключение заголовочного файла Windows API

#include <setupapi.h> // Подключение заголовочного файла для работы с Setup API

#include <cfgmgr32.h> // Подключение заголовочного файла для работы с Configuration Manager API

#include <usbiodef.h> // Подключение заголовочного файла для определений USB-устройств

#include <iostream> // Подключение стандартной библиотеки ввода-вывода

#include <vector> // Подключение библиотеки для работы с векторами

#include <string> // Подключение библиотеки для работы со строками

#include <iomanip> // Подключение библиотеки для форматирования вывода

#include <conio.h> // Подключение библиотеки для консольного ввода-вывода

#include <chrono> // Подключение библиотеки для работы со временем

#include <thread> // Подключение библиотеки для работы с потоками

#include <functional> // Подключение библиотеки для работы с функциональными объектами

std::wstring lastSafelyRemovedDeviceId; // Глобальная переменная для хранения ID последнего безопасно извлеченного устройства

struct Device { // Определение структуры Device для хранения информации об устройстве

std::wstring name; // Имя устройства

std::wstring hardwareId; // Аппаратный ID устройства

bool ejectable; // Флаг, указывающий, можно ли извлечь устройство

DEVINST devInst; // Дескриптор экземпляра устройства

Device() : ejectable(false), devInst(0) {} // Конструктор по умолчанию

void print() const { // Метод для вывода информации об устройстве

std::wcout << std::setw(40) << std::left << hardwareId << L" | " << name << std::endl;

}

[[nodiscard]] bool eject() const { // Метод для извлечения устройства

if (ejectable) { // Если устройство можно извлечь

return CM\_Request\_Device\_EjectW(devInst, nullptr, nullptr, 0, 0) == CR\_SUCCESS; // Запрос на извлечение устройства

}

return false; // Если устройство нельзя извлечь, возвращаем false

}

};

std::vector<Device> devices; // Глобальный вектор для хранения списка устройств

void updateDeviceList() { // Функция для обновления списка устройств

devices.clear(); // Очистка текущего списка устройств

HDEVINFO deviceInfo = SetupDiGetClassDevsW(&GUID\_DEVINTERFACE\_USB\_DEVICE, nullptr, nullptr, DIGCF\_PRESENT | DIGCF\_DEVICEINTERFACE); // Получение информации о классе USB-устройств

if (deviceInfo == INVALID\_HANDLE\_VALUE) return; // Если не удалось получить информацию, выходим из функции

SP\_DEVINFO\_DATA devInfoData; // Структура для хранения информации об устройстве

devInfoData.cbSize = sizeof(SP\_DEVINFO\_DATA); // Установка размера структуры

for (DWORD i = 0; SetupDiEnumDeviceInfo(deviceInfo, i, &devInfoData); i++) { // Перебор всех устройств

Device device; // Создание объекта устройства

device.devInst = devInfoData.DevInst; // Сохранение дескриптора экземпляра устройства

WCHAR buffer[256]; // Буфер для хранения строковых данных

if (SetupDiGetDeviceRegistryPropertyW(deviceInfo, &devInfoData, SPDRP\_DEVICEDESC, nullptr, reinterpret\_cast<PBYTE>(buffer), sizeof(buffer), nullptr)) {

device.name = buffer; // Получение и сохранение имени устройства

}

if (SetupDiGetDeviceRegistryPropertyW(deviceInfo, &devInfoData, SPDRP\_HARDWAREID, nullptr, reinterpret\_cast<PBYTE>(buffer), sizeof(buffer), nullptr)) {

device.hardwareId = buffer; // Получение и сохранение аппаратного ID устройства

}

DWORD properties; // Переменная для хранения свойств устройства

if (SetupDiGetDeviceRegistryPropertyW(deviceInfo, &devInfoData, SPDRP\_CAPABILITIES, nullptr, reinterpret\_cast<PBYTE>(&properties), sizeof(DWORD), nullptr)) {

device.ejectable = (properties & CM\_DEVCAP\_REMOVABLE) != 0; // Определение, является ли устройство извлекаемым

}

devices.push\_back(device); // Добавление устройства в список

}

SetupDiDestroyDeviceInfoList(deviceInfo); // Освобождение ресурсов, связанных с информацией об устройствах

}

void printDevices() { // Функция для вывода списка устройств

system("cls"); // Очистка консоли

std::wcout << L"Список USB-устройств:\n"; // Вывод заголовка

for (size\_t i = 0; i < devices.size(); i++) { // Перебор всех устройств

std::wcout << i + 1 << L". "; // Вывод номера устройства

devices[i].print(); // Вывод информации об устройстве

}

std::wcout << L"\nВыберите номер устройства для извлечения или 'q' для выхода.\n"; // Вывод инструкции

}

void checkForUnsafeRemoval() { // Функция для проверки небезопасного извлечения устройств

static std::vector<Device> previousDevices; // Статический вектор для хранения предыдущего списка устройств

static std::vector<std::wstring> reportedDevices; // Статический вектор для хранения ID устройств, о которых уже сообщалось

std::vector<Device> currentDevices = devices; // Копирование текущего списка устройств

// Проверяем устройства, которые были удалены

for (const auto& prevDevice : previousDevices) { // Перебор предыдущего списка устройств

auto it = std::find\_if(currentDevices.begin(), currentDevices.end(),

[&prevDevice](const Device& dev) { return dev.hardwareId == prevDevice.hardwareId; }); // Поиск устройства в текущем списке

if (it == currentDevices.end() && // Если устройство не найдено в текущем списке

std::find(reportedDevices.begin(), reportedDevices.end(), prevDevice.hardwareId) == reportedDevices.end() && // И о нем еще не сообщалось

prevDevice.hardwareId != lastSafelyRemovedDeviceId) { // И оно не было безопасно извлечено

std::wcout << L"Внимание: Устройство было извлечено небезопасно: " << prevDevice.name << L"\n"; // Вывод предупреждения

reportedDevices.push\_back(prevDevice.hardwareId); // Добавление устройства в список сообщенных

}

}

// Очистка списка сообщенных устройств, которые снова появились

reportedDevices.erase(

std::remove\_if(reportedDevices.begin(), reportedDevices.end(),

[&currentDevices](const std::wstring& reportedId) {

return std::any\_of(currentDevices.begin(), currentDevices.end(),

[&reportedId](const Device& dev) { return dev.hardwareId == reportedId; });

}),

reportedDevices.end()

);

// Обновляем список предыдущих устройств

previousDevices = currentDevices;

// Сбрасываем ID последнего безопасно извлеченного устройства

lastSafelyRemovedDeviceId.clear();

}

int main() { // Главная функция программы

setlocale(LC\_ALL, ""); // Установка локали для корректного отображения русских символов

bool running = true; // Флаг работы программы

auto lastUpdateTime = std::chrono::steady\_clock::now(); // Время последнего обновления списка устройств

while (running) { // Основной цикл программы

auto currentTime = std::chrono::steady\_clock::now(); // Текущее время

if (std::chrono::duration\_cast<std::chrono::seconds>(currentTime - lastUpdateTime).count() >= 1) { // Если прошла 1 секунда с последнего обновления

updateDeviceList(); // Обновление списка устройств

printDevices(); // Вывод списка устройств

checkForUnsafeRemoval(); // Проверка на небезопасное извлечение

lastUpdateTime = currentTime; // Обновление времени последнего обновления

}

if (\_kbhit()) { // Если нажата клавиша

int ch = \_getch(); // Получение кода нажатой клавиши

if (ch == 'q' || ch == 'Q') { // Если нажата 'q' или 'Q'

running = false; // Завершение работы программы

} else if (ch >= '1' && ch <= '9') { // Если нажата цифра от 1 до 9

int index = ch - '1'; // Вычисление индекса устройства

if (index < devices.size()) { // Если индекс в пределах списка устройств

if (devices[index].eject()) { // Попытка извлечения устройства

std::wcout << L"Устройство успешно извлечено\n"; // Сообщение об успешном извлечении

lastSafelyRemovedDeviceId = devices[index].hardwareId; // Сохранение ID безопасно извлеченного устройства

} else {

std::wcout << L"Не удалось извлечь устройство\n"; // Сообщение о неудачном извлечении

}

std::this\_thread::sleep\_for(std::chrono::seconds(3)); // Пауза на 3 секунды

updateDeviceList(); // Обновление списка устройств

printDevices(); // Вывод обновленного списка устройств

}

}

}

std::this\_thread::sleep\_for(std::chrono::milliseconds(100)); // Пауза в 100 миллисекунд для снижения нагрузки на процессор

}

return 0; // Завершение программы

}