ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОЙЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра вычислительных машин, комплексов, сетей и систем.

КУРСОВАЯ РАБОТА
по дисциплине «Операционные системы»
Тема: «Взаимодействие с операционной системой
с помощью внешних устройств»

Выполнил студент
2 курса
факультета ПМиВТ
группы ЭВМ 2–1
Лисовец Ярослав Васильевич
Преподаватель: Доцент Черкасова Н.И.

2022

Оглавление

Аннотация

В рамках курсовой работы было разработано приложение, позволяющее взаимодействовать с внешним устройством посредствам bluetooth. В качестве устройства выступает геймпад для Nintendo® Wii® - Wiimote.

Приложение способно установить соединение с геймпадом через bluetooth и считывать и передавать данные через HID.

Отчет по результатам работы состоит из графической части и пояснительной записки.

Пояснительная записка включает в себя техническое задание, структуру программы, информацию о системных требованиях и руководство пользователя. Графическая часть включает в себя листинги программ и результаты выполнения программы.

Техническое задание

Разработать программу демонстрирующую применение внешних интерфейсов для взаимодействия с операционной системой. Требования к возможностям программы

- Возможность установки соединения через bluetooth.
- Подключение к устройству через HID[1].
- Получение данных с устройства и их обработка.
- Передача команд на устройство.

Разработать отчет о проделанной работе, с использованием пособия[2]

Краткие теоретические сведения

2.1 Bluetooth

Bluetooth[3] — это стандарт беспроводной технологии малого радиуса действия, который используется для обмена данными между стационарными и мобильными устройствами на коротких расстояниях с использованием радиоволн УВЧ в диапазонах ISM от 2,402 до 2,48 ГГц. Он в основном используется в качестве альтернативы проводным соединениям, для обмена файлами между соседними портативными устройствами и подключения мобильных телефонов и музыкальных плееров с беспроводными наушниками. В наиболее широко используемом режиме мощность передачи ограничена 2,5 мВт, что обеспечивает очень малую дальность до 10 метров

2.2 Winsock2

Для работы с внешними интерфейсами, в частности с bluetooth[4], использовался Winsock2[5].

Windows Sockets 2 $Winsock$ позволяет программистам создавать расширенные приложения Интернета, интернет сети и другие сетевые приложения для передачи данных приложений по сети независимо от используемого сетевого протокола.

2.3 HID(Human Interface Device)

Для получения и передачи данных использовался HID
Устройства с HID-интерфейсом — это определение класса устройств. Использует универсальный драйвер USB для поддержки устройств HID, таких, как клавиатуры, мыши, игровые контроллеры и т.д. До HID[6] устройства могли использовать только строго определенные протоколы для мышей и клавиатуры.

Для внедрения оборудования требуется либо перегрузить данные в существующий протокол, либо создать нестандартное оборудование с собственным специализированным драйвером. HID обеспечивает поддержку этих устройств ”режима загрузки добавляя поддержку инновационных инноваций с помощью расширяемых, стандартизированных и легко программируемых интерфейсов.

2.4 Разработка с использованием продукции JetBrains и инструментов CMake

JetBrains

JetBrains[7] - компания созданная тремя русскими разработчиками. В данный момент специализирующаяся на создании инструментов для разработчиков программного обеспечения Среди её IDE использовались Clion и IntelliJ Rust плагин.

Clion

Clion[8] - это IDE предназначенная для разработки на языках C или C++. В неё включён пакте анализа кода, широкие возможности по генерации кода и перехода по нему в один клик. Clion понимает современные стандарты C++ и обеспечивает поддержку предпроцессора.

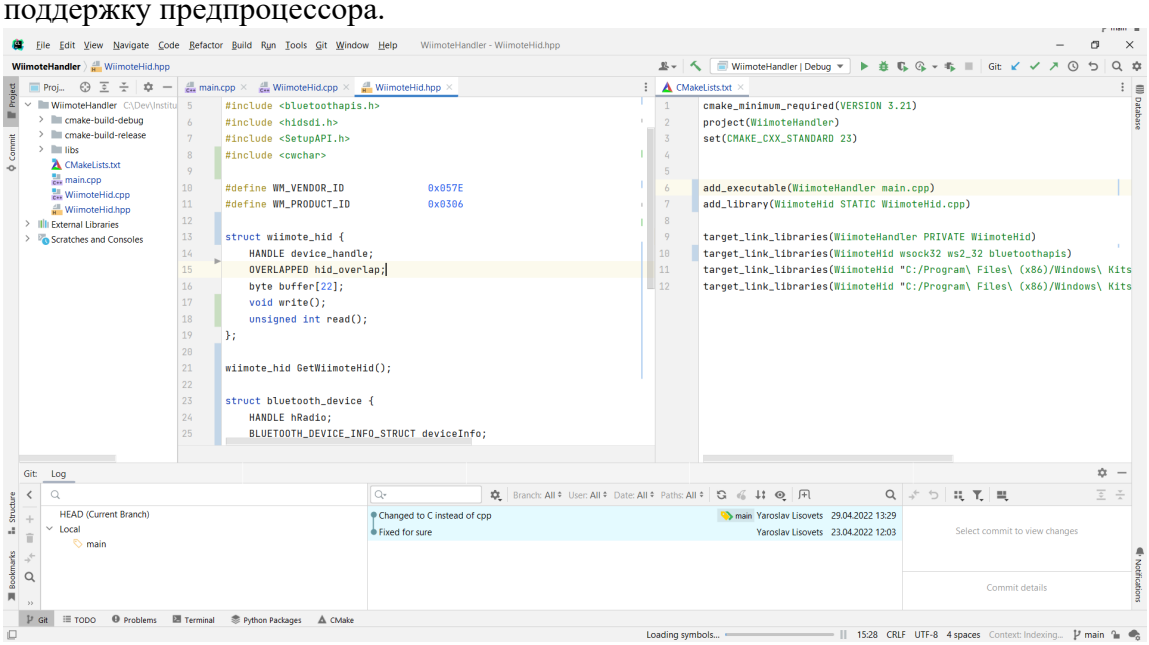

Рисунок - 1 Скриншот из Clion с открытыми окнами редактирования заголовочного файла и файла CMake

IntelliJ Rust

IntelliJ Rust[9] - плагин с открытым исходным кодом совместимый со всеми IDE основанными на IntelliJ IDEA. В паре с IntelliJ TOML [10] он направлен на то, чтобы привнести полный опыт IDE в ваш рабочий процесс с Rust и Cargo.

CMake

CMake[11] - это семейство инструментов с открытым исходным кодом для создания, тестирования программного обеспечения. CMake используется для управления процессом компиляции программного обеспечения с помощью простых файлов конфигурации, независимых от платформы и компилятора.

Создание приложения

3.1 Состав и Характеристики файлов проекта

Заголовочные файлы	
WiimoteHid.hpp	Основной заголовочный файл проекта, подключающий внешние заголовочные файлы и необходимый для работы с библиотекой. Также в нём происходит объявление функций библиотеки.
Файлы исходного кода	
WiimoteHid.cpp	Содержит реализацию функций объявленных в WiimoteHid.hpp
lib.rs	Является абстракцией над библиотекой для предоставления упрощённого интерфейса взаимодействия с геймпадом Wiimote
Вспомогательные файлы	
bindings.rs	Сгенерированные из WiimoteHid.hpp с помощью инструмента bindgen. Необходим для компиляции lib.rs.

Таблица 1 Описание файлов проекта

3.2 Стандартные классы и функции приложения

Название	Назначение	Параметры
BluetoothFindFirstRadio	Получение handle'а к bluetooth модулю.	Принимает указатель на структуру с параметрами, по которым будет вестись поиск, а так же указатель на handle.
BluetoothFindNextRadio	Возвращает handle к следующему bluetooth модулю, если тот имеется.	Принимает handle на предыдущий модуль, и handle куда будет записан новый в случае нахождения.
BluetoothGetRadioInfo	Получение данных таких, как адрес, имя, класс устройства и информации производителя, о bluetooth модуле.	Принимает структуру, в которую будет записана информация, и handle к модулю.
BluetoothFindFirstDevice	Получение handle'а первого найденного устройства в зоне действия bluetooth модуля.	Принимает структуру, куда будет записана информация о найденном устройстве, и параметрами, по которым будет вестись поиск.

BluetoothFindNextDevice	Получение handle'а следующего найденного устройства.	Принимает структуру, куда будет записана информация о найденном устройстве, и параметрами, по которым будет вестись поиск.
CloseHandle	Закрывает(возвращает системе) handle.	Принимает handle, который планируется закрыть.
wcscmp	Возвращает разницу между переданными строками.	Принимает два указателя на wchar_t
BluetoothRemoveDevice	Удаляет устройство из списка прежде подключённых в операционной.	Принимает адрес устройства, которое надо удалить.
BluetoothSetServiceState	Устанавливает состояние bluetooth сервиса Windows, например для указания сопряжения с устройством.	Принимает handle к bluetooth модулю, информации об устройстве, ID сервиса, через который будет происходить работа, устанавливаемое состояние.
HidD_GetHidGuid	Получение id для HID сервиса.	Принимает ссылку на переменную куда будет записан id.
SetupDiGetClassDevs	Функция возвращает дескриптор набора информации об устройстве, который содержит запрошенные элементы информации.	Принимает указатель на устройство, счётчик, handle к окну и флаги.
SetupDiEnumDeviceInterfaces	Функция SetupDiEnumDeviceInterfaces перечисляет интерфейсы устройств, которые содержатся в наборе информации об устройстве.	Принимает DeviceInfoSet, DeviceInfoData, ID устройства, индекс, переменную куда будут записаны данные.
HidD_GetAttributes	Возвращает атрибуты переданного устройства.	Получает handle устройства, и ссылку на структуру, куда будут записаны дынные.

CreateEvent	Служит для создания объекта события.	Получает указатель на структуру SECURITY_ATTRIBUTES, булевую переменную устанавливающую автоматический или ручной перезапуск, булевое начальное состояние, имя события.
ResetEvent	Перезапускает событие.	Получает ссылку на событие.
WriteFile	Записывает данные в указанный файл или устройство ввода/вывода (I/O).	Получает handle куда будет производиться запись, указатель на записываемые данные, количество записываемых байтов, ссылка на переменную куда будет записано количество отправленных байтов, указатель на переменную, которая содержит информацию, используемую при асинхронном (или перекрывающемся) вводе и выводе.
ReadFile	Считывает данные из указанного файла или устройства ввода/вывода (I/O).	Получает handle устройства/файла, указатель на буфер куда будут прочитаны данные, количество считываемых байтов, ссылка на переменную куда будет записано количество полученных байтов, указатель на переменную, которая содержит информацию, используемую при асинхронном (или перекрывающемся) вводе и выводе.
WaitForSingleObject	Ожидает событие в течение указанного времени.	Получает указатель на событие.

CancelIo	Отменяет ожидания ввода и вывода для устройства/файла.	Получает дескриптор устройства.
GetOverlappedResult	Извлекает результаты перекрывающейся/асинхронной операции с указанным файлом, или устройством, или именованным каналом.	Получает дескриптор устройства, ссылку на OVERLAPPED, ссылку на переменную куда будет записано количество прочитанных байтов, время ожидания.

Таблица 2 Таблица с описанием стандартных использованных функций

3.3 Пользовательские структуры в приложении

struct wiimote_hid

Необходима для хранения:
Hid дескриптора, через который происходит обращение к устройству
Переменной типа OVERLAPPED, предназначенной синхронизации с потоками ввода и вывода
Буфера, используемого при чтении и записи.
Через данную структуру происходит получение и отправка данных. Для этого содержит методы чтения и записи.

struct bluetooth_device

В данной структуре хранится HANDLE к bluetooth модулю и BLUETOOTH_DEVICE_INFO_STRUCT, с помощью которого происходит обращение к устройству.

3.4 Пользовательские функции в приложении

wiimote_hid::write()

Предназначена для записи данных из буфера структуры wiimote_hid.

wiimote_hid::read()

Предназначена для записи данных в буфер структуры wiimote_hid. Возвращает количество прочитанных байтов.

GetWiimoteHid()

Производит поиск геймпада среди подключённых hid устройств и при нахождении устанавливает с ним соединение.
Возвращает структуру wiimote_hid.

void ProcessWiimotes(bool new_scan, const T &callback = nullptr)

Производит поиск bluetooth устройств. При нахождении такого производит вызов callback функцию, устанавливающую соединение.

bool AttachWiimote(HANDLE hRadio, const BLUETOOTH_RADIO_INFO &radio_info, BLUETOOTH_DEVICE_INFO_STRUCT &deviceInfo)

Устанавливает соединение с геймпадом.

bool ForgetWiimote(BLUETOOTH_DEVICE_INFO_STRUCT &deviceInfo)

Производит удаление удаление устройства из списка операционной системы с ранее подключёнными устройствами.

void wiimoteDisconnect(bluetooth_device *bluetoothDevice)

Разрывает bluetooth соединение с геймпадом.

bluetooth_device *FindConnectWiimoteBLE()

Производит цикл поиска и подключения к устройству и возвращает структуру с дескриптором к bluetooth модулю и информацией о bluetooth-устройстве.

3.5 Пользовательские функции в приложении

wiimote_hid::write	Предназначена для записи данных из буфера структуры wiimote_hid.
wiimote_hid::read	Предназначена для записи данных в буфер структуры wiimote_hid.Возвращает количество прочитанных байтов.
GetWiimoteHid	Производит поиск геймпада среди подключённых hid устройств и при нахождении устанавливает с ним соединение. Возвращает структуру wiimote_hid.
void ProcessWiimotes	Производит поиск bluetooth устройств.При нахождении такого производит вызов callback функцию, устанавливающую соединение.
bool AttachWiimote	Устанавливает соединение с геймпадом.

bool ForgetWiimote	Производит удаление удаление устройства из списка операционной системы с ранее подключёнными устройствами.
void wiimoteDisconnect	Разрывает bluetooth соединение с геймпадом.
bluetooth_device *FindConnectWiimoteBLE	Производит цикл поиска и подключения к устройству и возвращает структуру с дескриптором к bluetooth модулю и информацией о bluetooth-устройстве.

3.6 Структура программы

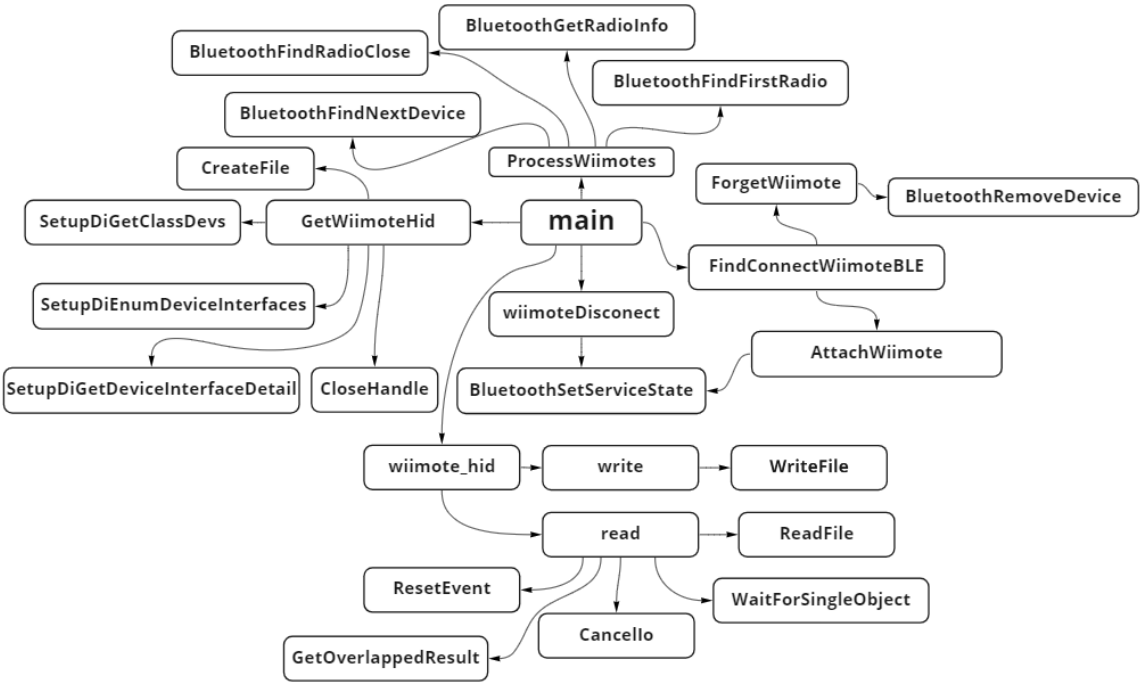

Рисунок - 2 Структура программы

3.7 Системные требования

1. Семейство операционных систем Windows.
2. Наличие свободного места в размере 158кб.
3. Bluetooth модуль.
4. Геймпад от Nintendo Wii - Wiimote.

Руководство по использованию

Для начала работы необходимо запустить программу. Для этого не обходимо нажать два раза по иконке приложения или выделить и уже после этого нажать клавишу enter.

После запуска в консоли появится сообщение о необходимости нажатия кнопок 1 и 2 для перевода геймпада в режим синхронизации. Также есть альтернатива в виде нажатия отельной кнопки синхронизации в отсеке с аккумуляторами. После чего должна пройти синхронизация. В случае успешного установления синхронизации будет выведено сообщение, оповещающее пользователя об этом. Совместно с этим сообщением будет выведена инструкция о работе с геймпадом.

Инструкция:

- Нажимайте кнопки 1 или 2 для изменения состояния светодиодного индикатора.
- Нажмите B для включения/отключения вибрации.
- Нажмите A, чтобы включить/отключить акселерометр.
- Одновременно нажмите A и Минус, чтобы отключить геймпад.

Рисунок - 3 Пример переключения светодиодов с помощью клавиш 1 и 2

```
Нажмите кнопки 1 и 2 одновременно или кнопку синхронизации в отсеке с аккумуляторами.
Геймпад подключён
Нажимайте кнопки 1 или 2 для изменения состояния светодиодного индикатора
Нажмите B для включения/отключения вибрации
Нажмите A, чтобы включить/отключить акселерометр
Нажмите вверх на DPAD'e, чтобы получить информацию о разработчике и приложении
Одновременно нажмите A и Минус, чтобы отключить геймпад
0b0110000|0b0001000|0b0000000|0b10001001|0b1110001|0b10000001|0b0000000|0b0000000|0b0000000|
Данное приложение было разработано Лисовцом Я.В. Оно  позволяет взаимодействовать с геймпадом Wiimote.
0b0110000|0b0000000|0b0000000|0b10001001|0b1110001|0b10000001|0b0000000|0b0000000|0b0000000|
0b0110000|0b0000000|0b0001000|0b10001001|0b1110001|0b10000001|0b0000000|0b0000000|0b0000000|
```

Рисунок - 4 Пример работы программы

Список литературы

1. Human Interface Device HID Windows https://docs.microsoft.com/ru-ru/windows-hardware/drivers/hid/
2. Пособие по оформлению курсовых и дипломных проектов и работ для студентов специальности 2201 – М.: МГТУ ГА, 2002.
3. Bluetooth Technology Overviewhttps://www.bluetooth.com/learn-about-bluetooth/tech-overview/
4. Библиотека выступающая в качестве примера по подключению к Wiimote по bluetooth https://github.com/dolphin-emu/dolphin
5. Winsock2 https://docs.microsoft.com/ru-ru/windows/win32/winsock/windows-sockets-start-page-2
6. Библиотека выбранная в качестве примера использования HID для работы с Wiimote https://github.com/wiiuse/wiiuse
7. Документ рассказывающий о JetBrains и их продукции Jetbrains Corporation Overview PDF https://resources.jetbrains.com/storage/products/jetbrains/docs/corporate-overview/en-us/jetbrains_corporate_overview.pdf
8. IDE от JetBrains предназначенная для разработки на языках C или C++.https://www.jetbrains.com/clion/
9. плагин с открытым исходным кодом совместимый со всеми IDE основанными на IntelliJ IDEA для Rust.https://plugins.jetbrains.com/plugin/8182-rust
10. IntelliJ TOML https://plugins.jetbrains.com/plugin/8195-toml
11. Семейство инструментов CMake https://cmake.org/

Приложение 1. Исходный тексты программы

6.1 WiimoteHid.hpp

```
#ifndef WIIMOTEHANDLER_WIIMOTEHID_HPP
#define WIIMOTEHANDLER_WIIMOTEHID_HPP

#include <WinSock2.h>
#include <bluetoothapis.h>
#include <hidsdi.h>
#include <SetupAPI.h>
#include <cwchar>

#define WM_VENDOR_ID 0x057E
#define WM_PRODUCT_ID 0x0306

struct wiimote_hid {
    HANDLE device_handle;
    OVERLAPPED hid_overlap;
    byte buffer[22];
    void write();
    unsigned int read();
};

wiimote_hid GetWiimoteHid();

struct bluetooth_device {
    HANDLE hRadio;
    BLUETOOTH_DEVICE_INFO_STRUCT deviceInfo;
};

template<typename T>
void ProcessWiimotes(bool new_scan, const T &callback = nullptr);

bool ForgetWiimote(BLUETOOTH_DEVICE_INFO_STRUCT &deviceInfo);

bool AttachWiimote(HANDLE hRadio, const BLUETOOTH_RADIO_INFO &radio_info, BLUETOOTH_DEVICE_INFO_STRUCT &deviceInfo);

void wiimoteDisconnect(bluetooth_device *bluetoothDevice);

bluetooth_device *FindConnectWiimoteBLE();

#endif
```

6.2 WiimoteHid.cpp

```
#include "WiimoteHid.hpp"


void wiimote_hid::write() {
    DWORD bytes;
    WriteFile(this->device_handle, this->buffer, 22, &bytes,
              &this->hid_overlap);
}

unsigned int wiimote_hid::read() {
    DWORD b = 0, r;
    while (not b) {
        if (!ReadFile(this->device_handle, this->buffer, 22, &b, &this->hid_overlap)) {
            r = WaitForSingleObject(this->hid_overlap.hEvent, 10);
            if (r == WAIT_TIMEOUT) {
                CancelIo(this->device_handle);
```

```
                ResetEvent(this->hid_overlap.hEvent);
                b = 0;
                continue;
            } else if (r == WAIT_FAILED) {
                b = 0;
                continue;
            }
            if (!GetOverlappedResult(this->device_handle, &this->hid_overlap, &b, 0)) {
                b = 0;
                continue;
            }
            ResetEvent(this->hid_overlap.hEvent);
            continue;
        }
    }
    return b;
}

struct wiimote_hid GetWiimoteHid() {
    GUID device_id;
    HANDLE dev;
    HDEVINFO device_info;
    int index;
    DWORD len;
    SP_DEVICE_INTERFACE_DATA device_data;
    PSP_DEVICE_INTERFACE_DETAIL_DATA detail_data = nullptr;
    HIDD_ATTRIBUTES attr;
    device_data.cbSize = sizeof(device_data);
    index = 0;
    HidD_GetHidGuid(&device_id);
    device_info = SetupDiGetClassDevs(&device_id, nullptr, nullptr, (DIGCF_DEVICEINTERFACE | DIGCF_PRESENT));
    struct wiimote_hid wiimote{};
    for (;; index++) {
        if (detail_data)
            free(detail_data);
        if (!SetupDiEnumDeviceInterfaces(device_info, nullptr, &device_id,index, &device_data))
            break;
        SetupDiGetDeviceInterfaceDetail(device_info, &device_data,
                    nullptr, 0, &len, nullptr);
        detail_data = (SP_DEVICE_INTERFACE_DETAIL_DATA_A *) malloc(len);
        detail_data->cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);
        if (!SetupDiGetDeviceInterfaceDetail(device_info, &device_data,
                        detail_data, len, nullptr, nullptr))
            continue;
        dev = CreateFile(detail_data->DevicePath, (GENERIC_READ | GENERIC_WRITE),
                        (FILE_SHARE_READ | FILE_SHARE_WRITE), nullptr, OPEN_EXISTING, FILE_FLAG_OVERLAPPED,
                        nullptr);
        if (dev == INVALID_HANDLE_VALUE)
            continue;
        attr.Size = sizeof(attr);
        HidD_GetAttributes(dev, &attr);
        if ((attr.VendorID == WM_VENDOR_ID)
            && ((attr.ProductID == WM_PRODUCT_ID))) {
            wiimote.device_handle = dev;
            wiimote.hid_overlap.hEvent = CreateEvent(nullptr, 1, 1, "");
            wiimote.hid_overlap.Offset = 0;
            wiimote.hid_overlap.OffsetHigh = 0;
            break;
        } else
            CloseHandle(dev);
    }
    return wiimote;
```

```
}

template<typename T>
void ProcessWiimotes(bool new_scan, const T &callback) {
    BLUETOOTH_DEVICE_SEARCH_PARAMS searchParams;
    searchParams.dwSize = sizeof(searchParams);
    searchParams.fReturnAuthenticated = true;
    searchParams.fReturnRemembered = true;
    searchParams.fReturnConnected = true;
    searchParams.fReturnUnknown = true;
    searchParams.fIssueInquiry = new_scan;
    searchParams.cTimeoutMultiplier = 1;

    BLUETOOTH_FIND_RADIO_PARAMS radioParam;
    radioParam.dwSize = sizeof(radioParam);
    bool found = false;
    HANDLE hRadio;

    HBLUETOOTH_RADIO_FIND hFindRadio = BluetoothFindFirstRadio(&radioParam, &hRadio);
    while (hFindRadio) {
        BLUETOOTH_RADIO_INFO radioInfo;
        radioInfo.dwSize = sizeof(radioInfo);

        auto const radioInfoResults = BluetoothGetRadioInfo(hRadio, &radioInfo);
        if (ERROR_SUCCESS == radioInfoResults) {
            searchParams.hRadio = hRadio;

            BLUETOOTH_DEVICE_INFO deviceInfo;
            deviceInfo.dwSize = sizeof(deviceInfo);
            auto wiimote_name = L"Nintendo RVL−CNT−01";
            // Enumerate BT devices
            do {
                HBLUETOOTH_DEVICE_FIND hFindDevice = BluetoothFindFirstDevice(&searchParams, &deviceInfo);
                do {
                    if (std::wcscmp(wiimote_name, deviceInfo.szName) == 0) {
                        if (deviceInfo.fConnected)
                            return;
                        callback(hRadio, radioInfo, deviceInfo);
                        found = true;
                    }
                } while (BluetoothFindNextDevice(hFindDevice, &deviceInfo));
            } while (not found);
        }

        if (false == BluetoothFindNextRadio(hFindRadio, &hRadio)) {
            CloseHandle(hRadio);
            BluetoothFindRadioClose(hFindRadio);
            hFindRadio = nullptr;
        }
    }
}

bool ForgetWiimote(BLUETOOTH_DEVICE_INFO_STRUCT &deviceInfo) {
    if (!deviceInfo.fConnected && deviceInfo.fRemembered) {
        BluetoothRemoveDevice(&deviceInfo.Address);
        deviceInfo.fRemembered = 0;
        return true;
    }
    return false;
}

bool AttachWiimote(HANDLE hRadio, const BLUETOOTH_RADIO_INFO &radio_info, BLUETOOTH_DEVICE_INFO_STRUCT &deviceInfo)
```

```
    ↪ {
    if (!deviceInfo.fConnected && !deviceInfo.fRemembered) {
        const DWORD hr = BluetoothSetServiceState(
                hRadio, &deviceInfo, &HumanInterfaceDeviceServiceClass_UUID, BLUETOOTH_SERVICE_ENABLE);
        if (FAILED(hr))
            return false;
        else
            return true;
    }
    return false;
}

void wiimoteDisconnect(bluetooth_device *bluetoothDevice) {
    BluetoothRemoveDevice(&bluetoothDevice->deviceInfo.Address);
}

bluetooth_device *FindConnectWiimoteBLE() {
    bluetooth_device *device;
    ProcessWiimotes(true, [&device](HANDLE hRadio, const BLUETOOTH_RADIO_INFO &radioInfo,
                                    BLUETOOTH_DEVICE_INFO_STRUCT &deviceInfo) {
        ForgetWiimote(deviceInfo);
        AttachWiimote(hRadio, radioInfo, deviceInfo);
        device = new bluetooth_device{hRadio, deviceInfo};
    });
    return device;
}
```

6.3 main.rs

```
use wiimotelib;
use wiimotelib::{button_states, wiimote_states, Buttons, Wiimote};
use wiimotelib::button_states::{A, B, MINUS, ONE, TWO, UP};
use wiimotelib::leds_states::{LED_1, LED_2, LED_3, LED_4};
use wiimotelib::wiimote_states::{ACCELEROMETER_ON, RUMBLE_ON};

fn main() {
    println!("Нажмите кнопки1 и2 одновременноиликлопкусинхронизацииводсекесаккумуляторами.");
    Wiimote::bleConnect();
    let message = "Нажимайте кнопки1 или2 дляизмененияcостояниясветодиодногоиндикатора\n\Нажмите
     B длявключенияотключения/ вибрации\n\Нажмите
     A, чтобывключитьотключить/ акселерометр\n\Нажмитевверхна
        DPADe', чтобыполучитьинформациюоразработчикеиприложении\n\Одновременнонажмите
      A иМинус, чтобыотключитьгеймпад";
    println!("Геймпад подключён");
    println!("{message}");
    let mut wiimote = Wiimote::find().unwrap();
    let mut vibration:bool=false;
    let mut led_number = 1u8;
    loop {
        wiimote.update();
        if wiimote.buttons.is_button_just_pressed(B) {
            vibration=!vibration;
            wiimote.set_vibration(!wiimote.get_state_value(RUMBLE_ON));
        }
        if wiimote.buttons.is_button_just_pressed(A){
            wiimote.set_accelerometer_state(!wiimote.get_state_value(ACCELEROMETER_ON));
        }
        if wiimote.buttons.is_button_just_pressed(ONE){
            led_number += 1;
            if led_number == 4{ led_number = 0;}
            wiimote.set_by_number(led_number);
        }
```

```
        if wiimote.buttons.is_button_just_pressed(TWO){
            led_number −= 1;
            if led_number == 0{ led_number = 4;}
            wiimote.set_by_number(led_number);
        }
        if wiimote.buttons.is_button_pressed(A)&&wiimote.buttons.is_button_pressed(MINUS){
            Wiimote::bleDisconnect();
            break;
        }
        if wiimote.buttons.is_button_pressed(UP){
            println!(”Данное приложениебылоразработаноЛисовцомЯВ.. Онопозволяетвзаимодействоватьсгеймпадом Wiimote.”)
        }
    }
}
```

6.4 lib.rs

```
#![allow(non_upper_case_globals)]
#![allow(non_camel_case_types)]
#![allow(non_snake_case)]
include!(”./bindings.rs”);
use crate::leds_states::{LED_1, LED_2, LED_3, LED_4};
use crate::wiimote_states::{ACCELEROMETER_ON, RUMBLE_ON};


pub mod button_states {
    pub const TWO: u16 = 0x0001;
    pub const ONE: u16 = 0x0002;
    pub const B: u16 = 0x0004;
    pub const A: u16 = 0x0008;
    pub const MINUS: u16 = 0x00010;
    pub const HOME: u16 = 0x00080;
    pub const LEFT: u16 = 0x00100;
    pub const RIGHT: u16 = 0x00200;
    pub const DOWN: u16 = 0x00400;
    pub const UP: u16 = 0x00800;
    pub const PLUS: u16 = 0x01000;
}

pub mod wiimote_states {
    pub const ACCELEROMETER_ON: u8 = 0b10000000;
    pub const RUMBLE_ON: u8 = 0b01000000;
}

pub struct Buttons {
    pressed: u16,
    released: u16,
    just_pressed: u16,
}


impl Buttons {
    fn new() −> Buttons {
        Buttons { pressed: 0, released: 0, just_pressed: 0 }
    }

    fn update(&mut self, buf: &[u8]) {
        let mut pressed = 0u16;
        pressed = pressed | (buf[1] as u16);
        pressed = pressed << 8;
        pressed = pressed | (buf[2] as u16);
        self.released = (!pressed) & self.pressed;
```

```
            self.just_pressed = (!self.released) & self.pressed ^ pressed;
            self.pressed = pressed;
            // println!("pressed: {:#016b}", pressed);
            // println!("released: {:#016b}", self.just_pressed);
            for i in &buf[..9] {
                print!("{i:#09b}|");
            }
            print!("\n")
        }
        pub fn is_button_pressed(&self, button: u16) -> bool {
            self.pressed & button != 0
        }
        pub fn is_button_just_pressed(&self, button: u16) -> bool {
            self.just_pressed & button != 0
        }
        pub fn is_button_released(&self, button: u16) -> bool {
            self.released & button != 0
        }
}


pub mod leds_states {
        pub const LED_1: u8 = 0x10;
        pub const LED_2: u8 = 0x20;
        pub const LED_3: u8 = 0x40;
        pub const LED_4: u8 = 0x80;
}

pub struct Wiimote {
        state: u8,
        led_state: u8,
        device: wiimote_hid,
        pub buttons: Buttons,
        accelerometer: [i16; 3],
        accelerometer_initial: [u8; 3],
        buffer: [byte; 22usize],
}

static mut ble_device: Option<*mut bluetooth_device> = None;

impl Wiimote {
        pub fn new() -> Wiimote {
            unsafe {
                let wiimote = GetWiimoteHid();
                Wiimote {
                    state: 0,
                    led_state: 0,
                    device: wiimote,
                    buttons: Buttons::new(),
                    accelerometer: [0; 3],
                    accelerometer_initial: [0; 3],
                    buffer: [0; 22usize],
                }
            }
        }

        pub fn find() -> Option<Wiimote> {
            Some(Wiimote::new())
        }

        pub fn bleConnect() {
            unsafe { ble_device = Some(FindConnectWiimoteBLE()); }
```

```
}

pub fn bleDisconnect() {
    unsafe { wiimoteDisconect(ble_device.unwrap()); }
}
//
pub fn write(&mut self, len: usize) {
    unsafe {
        self.device.buffer = [0u8; 22];
        for n in 0..len + 1 {
            self.device.buffer[n] = self.buffer[n];
        }
        self.device.write();
    }
}

//Function will write lat report in buffer
pub fn sync(&mut self) {
    unsafe {
        let len: usize = self.device.read().try_into().unwrap();
        for n in 0..len {
            self.buffer[n] = self.device.buffer[n];
        }
    }
}
pub fn update(&mut self) {
    self.sync();
    self.buttons.update(&self.device.buffer[..]);
    if self.get_state_value(ACCELEROMETER_ON) {}
}
pub fn set_vibration(&mut self, working: bool) {
    self.state &= !RUMBLE_ON;
    self.state |= RUMBLE_ON * (working as u8);
    self.buffer[0] = 0x11;
    self.buffer[1] = working as u8;
    self.write(3);
}
pub fn set_accelerometer_state(&mut self, working: bool) {
    self.state &= !ACCELEROMETER_ON;
    self.state |= ACCELEROMETER_ON * (working as u8);
    self.buffer = [0; 22];
    self.buffer[0] = 0x12;
    self.buffer[2] = 0x30 | working as u8;
    self.write(3);
}


pub fn get_state_value(&self, state: u8) -> bool {
    self.state & state != 0
}

pub fn set_state_value(&mut self, state_type: u8, value: bool) {
    self.state |= state_type * (value as u8);
}

pub fn set_leds(&mut self, value: u8) {
    self.led_state = value;
    self.buffer[0] = 0x11;
    self.buffer[1] = value;
    self.write(2);
}
```

```
    pub fn set_by_number(&mut self, number: u8) {
        let led = match number {
            1 => { LED_1 }
            2 => { LED_2 }
            3 => { LED_3 }
            _ => LED_4
        };
        self.set_leds(led);
    }
}
```

Приложение 2. Алгоритмы функций

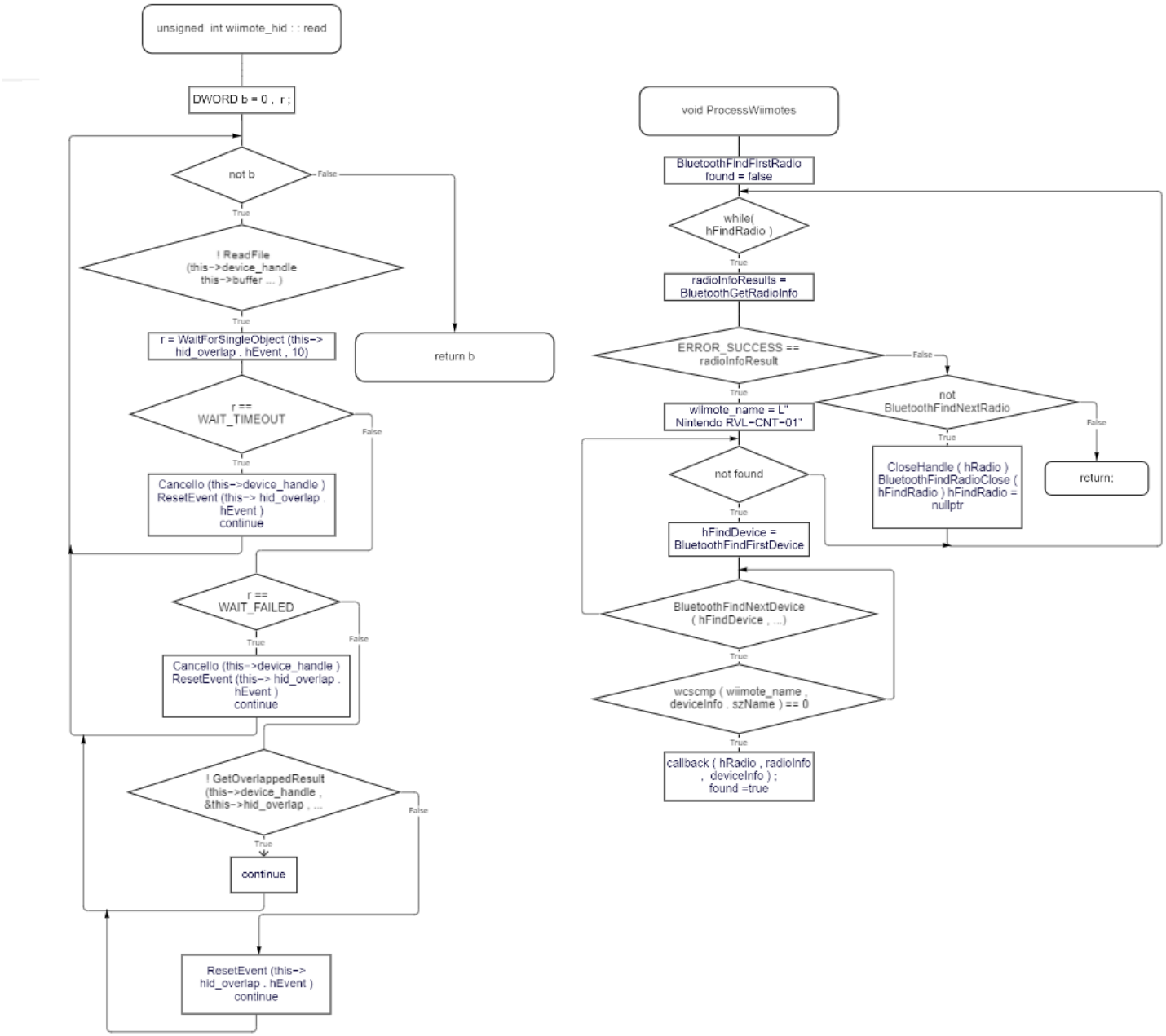

Рисунок - 5 Алгоритмы функций wiimote::read и ProcessWiimotes