МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
 РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

«НИЗКОУРОВНЕВАЯ РАБОТА С ПЕРИФЕРИЙНЫМИ УСТРОЙСТВАМИ»

студента 2 курса, группы 23201  
  
Смирнова Гордея Андреевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Матвеев Алексей Сергеевич

Новосибирск 2024

# Цель

Ознакомиться с началами низкоуровневого программирования периферийных устройств на примере получения информации о доступных USB-устройствах с помощью библиотеки libusb.

# Задание

Изучить принципы низкоуровневой работы с USB-устройствами с помощью библиотеки libusb. Ознакомиться с тем, как происходит работа с устройствами и какие данные хранит само устройство. Проанализировать работу кода и сделать выводы на основе результата работы.

# Дополнительное задание

Написать с использованием библиотеки libusb программу-драйвер для USB-мыши, выводящую все полученные данные от периферийного устройства.

# Листинг программы (список устройств)

#include <iostream>

#include <libusb.h>

typedef unsigned char uchar;

void printDeviceInfo(libusb\_device \*dev) {

libusb\_device\_descriptor desc;

int code = libusb\_get\_device\_descriptor(dev, &desc);

if (code < 0) {

std::cerr << "Error: failed to get device descriptor, code: " << code << std::endl;

return;

}

std::cout << "Class: " << static\_cast<int>(desc.bDeviceClass) << std::endl;

std::cout << "Vendor ID: " << desc.idVendor << std::endl;

std::cout << "Product ID: " << desc.idProduct << std::endl;

libusb\_device\_handle \*handle;

code = libusb\_open(dev, &handle);

if (code == 0) {

if (uchar vendor[64];

libusb\_get\_string\_descriptor\_ascii(handle, desc.iManufacturer, vendor, sizeof(vendor))) {

std::cout << "Vendor: " << vendor << std::endl;

}

if (uchar product[64];

libusb\_get\_string\_descriptor\_ascii(handle, desc.iProduct, product, sizeof(product))) {

std::cout << "Product: " << product << std::endl;

}

if (uchar serialNumber[64];

libusb\_get\_string\_descriptor\_ascii(handle, desc.iSerialNumber, serialNumber, sizeof(serialNumber))) {

std::cout << "Serial number: " << serialNumber << std::endl;

}

libusb\_close(handle);

}

else std::cerr << "Error: failed to open device." << std::endl;

std::cout << "--------------------------" << std::endl;

}

int main() {

libusb\_context \*ctx = nullptr;

if (const int code = libusb\_init(&ctx); code < 0) {

std::cerr << "Error: failed to initialize libusb, code: " << code << std::endl;

return 1;

}

libusb\_device \*\*devs;

const ssize\_t cnt = libusb\_get\_device\_list(ctx, &devs);

if (cnt < 0) {

std::cerr << "Error: failed to get device list." << std::endl;

libusb\_exit(ctx);

return 1;

}

std::cout << "USB devices found: " << cnt << std::hex << std::endl;

std::cout << "==========================" << std::endl;

for (ssize\_t i = 0; i < cnt; i++) {

printDeviceInfo(devs[i]);

}

libusb\_free\_device\_list(devs, 1);

libusb\_exit(ctx);

return 0;

}

# Листинг программы (драйвер мыши)

#include <iostream>

#include <libusb-1.0/libusb.h>

#define VENDOR\_ID 0x1af3

#define PRODUCT\_ID 0x0001

class USBMouse {

public:

USBMouse() {

if (libusb\_init(&context) < 0) {

throw std::runtime\_error("Failed to initialize libusb");

}

handle = libusb\_open\_device\_with\_vid\_pid(context, VENDOR\_ID, PRODUCT\_ID);

if (!handle) {

libusb\_exit(context);

throw std::runtime\_error("Failed to open device");

}

libusb\_detach\_kernel\_driver(handle, 0);

if (libusb\_claim\_interface(handle, 0) < 0) {

std::cerr << libusb\_claim\_interface(handle, 0) << std::endl;

libusb\_close(handle);

libusb\_exit(context);

throw std::runtime\_error("Failed to claim interface");

}

}

~USBMouse() {

if (handle) {

libusb\_release\_interface(handle, 0);

libusb\_attach\_kernel\_driver(handle, 0);

libusb\_close(handle);

}

libusb\_exit(context);

}

void run() const {

while (true) {

unsigned char data[6];

int transferred;

int code = libusb\_interrupt\_transfer(handle, 0x81, data, sizeof(data), &transferred, 0);

if (code == 0) printMouseData(data);

else std::cerr << "Data transferring error: " << libusb\_error\_name(code) << std::endl;

if (data[0] & 0x01 && data[0] & 0x02 && data[0] & 0x04) {

std::cout << "LMB + MMB + RMB pressed. Stopping..." << std::endl;

break;

}

}

}

private:

libusb\_context \*context = nullptr;

libusb\_device\_handle \*handle = nullptr;

static void printMouseData(const unsigned char \*data) {

int x = data[1] - data[2]; // 1, 3 перемещение (если отрицательно то 255-значение)

int y = data[3] - data[4]; // 2, 4 принимают значения 0 и 255

int btn = data[0];

int scr = data[5];

if (btn & 0x01) std::cout << "LMB pressed. ";

if (btn & 0x02) std::cout << "RMB pressed. ";

if (btn & 0x04) std::cout << "MMB pressed. ";

if (btn & 0x08) std::cout << "Back pressed. ";

if (btn & 0x10) std::cout << "Forward pressed. ";

if (scr == 0x01) std::cout << "Scroll up. ";

if (scr == 0xff) std::cout << "Scroll down. ";

std::cout << "Move: (" << x << ", " << y << ")" << std::endl;

}

};

int main() {

try {

USBMouse mouse;

mouse.run();

} catch (const std::runtime\_error &e) {

std::cerr << "Error: " << e.what() << std::endl;

return EXIT\_FAILURE;

}

return EXIT\_SUCCESS;

}

# Описание обнаруженных USB-устройств

Программа, запущенная локально на ноутбуке Lenovo Thinkpad T14s Gen 1 (AMD), вывела следующие устройства:

# USB devices found: 10

==========================

Class: 9

Vendor ID: 1d6b

Product ID: 3

Vendor: Linux 6.12.4-arch1-1 xhci-hcd

Product: xHCI Host Controller

Serial number: 0000:06:00.4

// встроенный контроллер USB 3.0

--------------------------

Class: e0

Vendor ID: 8087

Product ID: 29

Vendor: 0000:06:00.4

Product: 0000:06:00.4

Serial number: 0000:06:00.4

// bluetooth-модуль Intel AX200

--------------------------

Class: 9

Vendor ID: 1d6b

Product ID: 2

Vendor: Linux 6.12.4-arch1-1 xhci-hcd

Product: xHCI Host Controller

Serial number: 0000:06:00.4

// встроенный контроллер USB 2.0

--------------------------

Class: 9

Vendor ID: 1d6b

Product ID: 3

Vendor: Linux 6.12.4-arch1-1 xhci-hcd

Product: xHCI Host Controller

Serial number: 0000:06:00.3

// встроенный контроллер USB 2.0

--------------------------

Class: 0

Vendor ID: 1af3

Product ID: 1

Vendor: Kingsis Peripherals

Product: ZOWIE Gaming mouse

Serial number: ZOWIE Gaming mouse

// подключенная мышь Zowie EC1b

--------------------------

Class: 9

Vendor ID: 1d6b

Product ID: 2

Vendor: Linux 6.12.4-arch1-1 xhci-hcd

Product: xHCI Host Controller

Serial number: 0000:06:00.3

// встроенный контроллер USB 2.0

--------------------------

Class: 9

Vendor ID: 1d6b

Product ID: 3

Vendor: Linux 6.12.4-arch1-1 xhci-hcd

Product: xHCI Host Controller

Serial number: 0000:05:00.0

// встроенный контроллер USB 3.0

--------------------------

Class: ef

Vendor ID: 13d3

Product ID: 5406

Vendor: Azurewave

Product: Integrated Camera

Serial number: 0000

// встроенная веб-камера ноутбука

--------------------------

Class: 9

Vendor ID: 1d6b

Product ID: 2

Vendor: Linux 6.12.4-arch1-1 xhci-hcd

Product: xHCI Host Controller

Serial number: 0000:05:00.0

// встроенный контроллер USB 2.0

--------------------------

Class: 9

Vendor ID: 1d6b

Product ID: 2

Vendor: Linux 6.12.4-arch1-1 ehci\_hcd

Product: EHCI Host Controller

Serial number: 0000:02:00.4

// встроенный контроллер USB 2.0

--------------------------

Опытным путём было выявлено, что контроллеры 3.0 отвечают за внешние USB-порты ноутбука, а контроллеры 2.0 — за встроенные периферийные устройства, такие как клавиатура, тачпад, трекпоинт, микрофон и т.д.

# Описание работы драйвера мыши

Во время работы программы драйвер мыши отсоединяется от ядра системы, чтобы программа могла получить к ней доступ. Таким образом, взаимодействие с мышью не приводит к работе с курсором, но влияет на выходные данные программы. По завершении работы программы (в нашем случае для этого нужно нажать ЛКМ+СКМ+ПКМ) драйвер мыши снова подсоединяется к ядру системы.

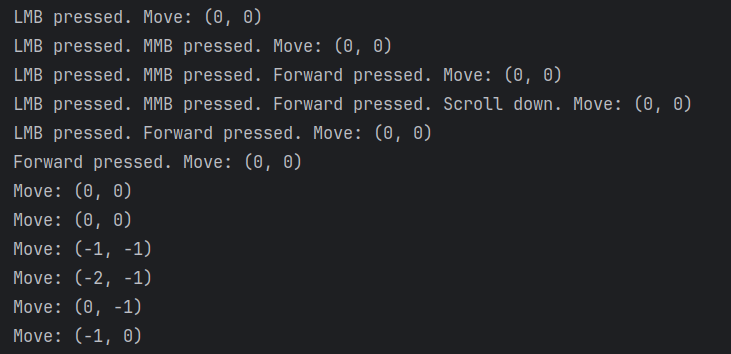
Обнаружение и подключение к устройству происходит по vendorID и productID, полученным в вышеописанной программе.

В случае конкретной мыши (Zowie EC1b) каждую операцию USB-опроса контроллер получает ровно 6 байт данных:

[0] — здесь задействованы 5 бит из 8, каждый из пяти принимает значение 1 если нажата соответствующая кнопка (ЛКМ, ПКМ, СКМ, боковая-назад, боковая-вперёд);  
[1], [3] — перемещение по оси X (вправо/влево). При движении вправо 3й байт принимает значение 0, а при движении влево — значение 255. При этом 1й байт при движении вправо принимает значение (0 + <модуль сдвига координат>), при движении влево (255 - <модуль сдвига координат>). Таким образом, в человекочитаемом виде формула сдвига имеет вид ([1] - [3]) и может принимать значения от -255 до 255.

[2], [4] — перемещение по оси Y (назад/вперед). Работает полностью аналогично предыдущему пункту.  
[5] — принимает значение 0 если колесо мыши не используется, значение 1 при скролле вверх, значение 255 при скролле вниз.  
 На основе полученных 6 байтов в консоль при каждом USB-опросе выводятся данные о взаимодействии с мышью. На данной модели мыши частоту опроса можно менять физическим переключателем от 125 до 1000Гц. Соответственно, на максимальной частоте данные выводятся в 8 раз чаще, чем на минимальной.

Пример вывода:



# 

# Заключение

В ходе работы с libusb были освоены:

1) работа с usb-устройствами;

2) получение данных с usb-устройств и оперирование над ними.