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 ПРАКТИЧЕСКОЙ РАБОТЫ

Практическая работа №6

НИЗКОУРОВНЕВАЯ РАБОТА С ПЕРИФЕРИЙНЫМИ УСТРОЙСТВАМИ

студента 2 курса, группы 23201

Сорокина Матвея Павловича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
А.С. Матвеев

Новосибирск 2024

СОДЕРЖАНИЕ

ЦЕЛЬ

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

ЗАДАНИЕ

1. Реализовать программу, получающую список всех подключенных к машине USB устройств с использованием libusb. Для каждого найденного устройства напечатать его класс, идентификатор производителя и идентификатор изделия.
2. Изучить состав и характеристики обнаруженных с помощью реализованной программ USB устройств.
3. Дополнить программу, реализованную ранее функцией печати серийного номера USB устройства. Для написания функции рекомендуется использовать функции *libusb_open*, *libusb_close*, *libusb_get_string_descriptor_ascii* для печати поля *iSerialNumber* дескриптора устройства.
4. Составить отчет по лабораторной работе. Отчет должен содержать следующее:
    - Титульный лист.
    - Цель лабораторной работы.
    - Полный компилируемый листинг реализованной программы и команды для ее компиляции.
    - Описание обнаруженных USB-устройств.
    - Вывод по результатам лабораторной работы.

ОПИСАНИЕ РАБОТЫ

Полный листинг программы и пример ее импользования предоставлены. (см. Приложения 1,2)

libusb — это библиотека для работы с USB (Universal Serial Bus) устройствами, которая предоставляет унифицированный интерфейс для взаимодействия с ними на низком уровне. Она позволяет разработчикам взаимодействовать с USB-устройствами напрямую из пользовательского кода, минуя более высокоуровневые системные API или драйверы. Основная особенность libusb заключается в том, что она работает кроссплатформенно, поддерживая Windows, macOS и Linux, что упрощает разработку приложений для различных операционных систем.

Рассмотрим работу программы *main.cpp*:

1. Подключение библиотек:
   - ***iostream*** - заголовочный файл с классами, функциями и переменными для организации ввода-вывода. Используется для записи в стандартный поток вывода *stdout*.
   - ***libusb.h*** - предоставляет множество функций, которые позволяют получить информацию об имеющихся в системе USB-устройствах.
2. Функция *main:*
   - Объявление переменных для работы с libusb:

```
libusb_device **devs;
libusb_context *ctx = NULL;
int r;
ssize_t cnt;
```

     - ***libusb_device **devs*** - указатель на указатель для хранения списка устройств.
     - ***libusb_context *ctx*** - указатель для хранения контекста *libusb*.
     - ***int r*** - переменная для хранения возвращаемого значения функции *libusb_init(&ctx)*

> int **libusb_init**(libusb_context **ctx)
> This function initialises libusb. It must be called at the beginning of the program, before other libusb routines are used. This function returns 0 on success or LIBUSB_ERROR on failure.

- ***ssize_t cnt*** - переменная для количества найденных USB-устройств.

- Инициализация *libusb:*

```
r = libusb_init(&ctx);
    if (r < 0) {
        std::cerr << "Ошибка: не удалось инициализировать libusb,
код: " << r << std::endl;
        return 1;
    }
```

  - Функция *libusb_init* инициализирует библиотеку и создаёт контекст сессии, сохраняя его в ***ctx***.
  - Если *libusb_init* возвращает отрицательное значение, выводится сообщение об ошибке, и программа завершается.

- Получение списка USB-устройств:

```
cnt = libusb_get_device_list(ctx, &devs);
if (cnt < 0) {
    std::cerr << "Ошибка: не удалось получить список устройств."
<< std::endl;
    libusb_exit(ctx);
    return 1;
    }
```

  - *libusb_get_device_list* получает список подключённых USB-устройств и сохраняет их в ***devs***.
  - Если функция возвращает отрицательное значение, выводится сообщение об ошибке, *libusb* завершает свою работу с помощью *libusb_exit*, и программа завершается.

> void **libusb_exit**(libusb_context *ctx)
>
> Deinitialise libusb. Must be called at the end of the application. Other libusb routines may not be called after this function.

  - Выводится количество найденных USB-устройств.

- Перебор всех найденных устройств и вывод их информации:

```
for (ssize_t i = 0; i < cnt; i++) {
    printDeviceInfo(devs[i]);
}
```

  - В цикле *for* каждое устройство из списка передаётся в функцию *printDeviceInfo* для вывода информации о нём.

- Освобождение ресурсов:

```
libusb_free_device_list(devs, 1);
libusb_exit(ctx);

return 0;
```

  - *libusb_free_device_list* освобождает память, выделенную под список устройств.

> void **libusb_free_device_list**(libusb_device **list, int unref_devices)
>
> Free the list of devices discovered by libusb_get_device_list. If *unref_device* is set to 1 all devices in the list have their reference counter decremented once

  - *libusb_exit* завершает работу с *libusb* и освобождает ресурсы контекста.
  - *return 0* завершает выполнение функции *main.*

3. Функция *printDeviceInfo:*
   - Параметр функции:
     - *libusb_device *dev* — указатель на устройство, информацию о котором нужно вывести.
   - Получение дескриптора устройства:

```
libusb_device_descriptor desc;
int r = libusb_get_device_descriptor(dev, &desc);
if (r < 0) {
    std::cerr << "Ошибка: не удалось получить дескриптор
устройства, код: " << r << std::endl;
    return;
}
```

     - Создаётся переменная *libusb_device_descriptor* ***desc*** для хранения дескриптора устройства. Затем вызывается функция *libusb_get_device_descriptor(dev, &desc)*, чтобы заполнить ***desc***.

> int **libusb_get_device_descriptor**(libusb_device *dev, libusb_device_descriptor *desc)
> Get the USB device descriptor for the device ***dev***. This is a non-blocking function. Returns 0 on success and a LIBUSB_ERROR code on failure.

- Если возвращается отрицательный результат, выводится сообщение об ошибке, и функция завершается.

- Вывод информации о классе устройства, идентификаторах производителя и изделия:

```
std::cout << "Класс устройства: " << (int)desc.bDeviceClass
<< std::endl;
std::cout << "Идентификатор производителя: " <<
(int)desc.idVendor << std::endl;
std::cout << "Идентификатор устройства: " << std::hex <<
desc.idProduct << std::endl;
```

- Получение и вывод серийного номера устройства:

```
if (desc.iSerialNumber) {
        libusb_device_handle *handle;
        r = libusb_open(dev, &handle);
        if (r == 0) {
            unsigned char serialNumber[256];
            int length =
libusb_get_string_descriptor_ascii(handle,
desc.iSerialNumber, serialNumber, sizeof(serialNumber));
            if (length > 0) {
                std::cout << "Серийный номер: " <<
serialNumber << std::endl;
            } else {
                std::cerr << "Ошибка: не удалось получить
серийный номер." << std::endl;
            }
            libusb_close(handle);
        } else {
            std::cerr << "Ошибка: не удалось открыть
устройство для получения серийного номера." << std::endl;
        }
    } else {
        std::cout << "Серийный номер отсутствует." <<
std::endl;
}
```

  - Проверяется, имеет ли устройство поле серийного номера (*desc.iSerialNumber*).
  - Если серийный номер есть, устройство открывается с помощью *libusb_open*, а дескриптор сохраняется в *handle*.

int **libusb_open**(libusb_device *dev, libusb_device_handle **devh)

Open a device and obtain a device_handle. **Returns 0 on success**, LIBUSB_ERROR_NO_MEM on memory allocation problems, LIBUSB_ERROR_ACCESS on permissions problems, LIBUSB_ERROR_NO_DEVICE if the device has been disconnected and a LIBUSB_ERROR code on other errors.

- Если *libusb_open* возвращает 0, создаётся буфер *serialNumber* для хранения серийного номера, и вызывается функция *libusb_get_string_descriptor_ascii*, которая пытается получить серийный номер в виде строки.

  Требует открытого *libusb_device_handle* (полученного с помощью ***libusb_open***), так как без открытого устройства невозможно получить доступ к этим строковым данным.

  int **libusb_get_string_descriptor_ascii**(libusb_device_handle *devh, uint8_t desc_idx, unsigned char *data, int length)

  Retrieve a string descriptor in C style ASCII. Returns the positive number of bytes in the resulting ASCII string on success and a LIBUSB_ERROR code on failure.

  - Если операция успешна, серийный номер выводится. В противном случае выводится ошибка.
  - Устройство закрывается функцией *libusb_close*.

    void **libusb_close**(libusb_device_handle *devh)

    Close a device handle.

  - Если устройство не удалось открыть, выводится сообщение об ошибке.
- Выводится строка для разделения информации о разных устройствах.

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

Теперь более подробно рассмотрим процессы, происходящие при выполнении некоторых программных функций:

libusb_context - структура, представляющая сессию работы с библиотекой *libusb*. Она хранит состояние библиотеки. Контекст нужен для изоляции нескольких независимых сессий, что позволяет, например, запустить параллельно несколько приложений, работающих с libusb, не влияя друг на друга.

Сессия - выделенный период работы программы с USB-устройствами, в рамках которого библиотека сохраняет свое состояние.
В течение этой сессии программа получает доступ к USB-устройствам, выполняет операции с данными этих устройств.

Также выжно понимать что происходит с переменной ***libusb_context *ctx*** типа:

- int **libusb_init**(libusb_context **ctx)
  Выделяет память и создает новый объект *libusb_context*, который хранит состояние библиотеки. Заполняет указатель ****ctx*** адресом нового контекста.
- void **libusb_exit**(libusb_context *ctx)
  Освобождает всю память и другие ресурсы, связанные с контекстом *libusb_context*. Контекст больше не может использоваться для взаимодействия с USB-устройствами.
- ssize_t **libusb_get_device_list**(libusb_context *ctx, libusb_device ***list)
  Функция принимает указатель на *libusb_context*, заполняет массив указателей на объекты *libusb_device*, представляющие все USB-устройства, подключенные к системе в данный момент. Она выделяет память для данного списка указателей. Функция возвращает количество найденных USB-устройств в списке.

Учитывая что в последней рассматриваемой функции ***libusb_get_device_list*** происходит выделение памяти на массив с указателями типа *libusb_device* мы осознаем, что данная память должна также освобождаться во избежание утечек памяти:

- void **libusb_free_device_list**(libusb_device **list, int unref_devices)
  Освобождает память, выделенную для массива указателей на объекты *libusb_device*.

Находясь в функции void **printDeviceInfo**(libusb_device *dev) вывод характеристики каждого элемента массива указателей ***libusb_device **devs***

происходит используя поля ***libusb_device_descriptor desc*** – это структура, содержащая информацио о USB-устройстве.
Поля этой структуры описаны здесь: https://libusb.sourceforge.io/api-1.0/structlibusb__device__descriptor.html

Когда вызываем ***libusb_open()***, то получаем указатель на объект *device_handle*. Он играет роль интерфейса для взаимодействия с USB-устройством, позволяя управлять его функциями (например, чтение/запись данных). Этот дескриптор необходим для вызова различных функций libusb, чтобы выполнить операции на уровне ввода-вывода, например, запрос данных.

После того как работа с устройством завершена, мы должны закрыть дескриптор с помощью ***libusb_close()***. Это освобождает ресурсы, связанные с этим дескриптором, и позволяет безопасно завершить взаимодействие с устройством.

Используемые в моей программе поля структуры *libusb_device_descriptor:*

1. ***bDeviceClass;***
2. ***idVendor;***
3. ***idProduct;***
4. ***iSerialNumber;***
5. ***iManufacturer;***
6. ***iProduct.***

Просмотреть полный список можно по ссылке:
https://libusb.sourceforge.io/api-1.0/structlibusb__device__descriptor.html

При выполнении программы мы получаем список подключенных к системе USB-устройств:

```
Найдено USB-устройств: 6
==========================
Класс устройства: 9
Идентификатор производителя: 7531
Идентификатор устройства: 3
Серийный номер: 0000:00:14.0
Производитель: Linux 6.8.0-44-generic xhci-hcd
Изделие: xHCI Host Controller
-------------------------
Класс устройства: ef
Идентификатор производителя: 3277
```

```
Идентификатор устройства: 4
Производитель: Sonix Technology Co., Ltd.
Изделие: Integrated Webcam_FHD
-------------------------
Класс устройства: e0
Идентификатор производителя: 8087
Идентификатор устройства: 33
-------------------------
Класс устройства: 9
Идентификатор производителя: 1d6b
Идентификатор устройства: 2
Серийный номер: 0000:00:14.0
Производитель: Linux 6.8.0-44-generic xhci-hcd
Изделие: xHCI Host Controller
-------------------------
Класс устройства: 9
Идентификатор производителя: 1d6b
Идентификатор устройства: 3
Серийный номер: 0000:00:0d.0
Производитель: Linux 6.8.0-44-generic xhci-hcd
Изделие: xHCI Host Controller
-------------------------
Класс устройства: 9
Идентификатор производителя: 1d6b
Идентификатор устройства: 2
Серийный номер: 0000:00:0d.0
Производитель: Linux 6.8.0-44-generic xhci-hcd
Изделие: xHCI Host Controller
```

Подробно с характеристикой классов устройств можно ознакомиться по ссылке: https://www.usb.org/defined-class-codes

Ниже представлено краткое описание полученных устройств:

1. Четыре ***xHCI Host Controller*** - контроллер обеспечивает обмен данными между центральным процессором (CPU) и устройствами на шине, обрабатывает команды от процессора и передаёт их устройству, а также принимает данные с шины и передаёт их процессору.

> According to the USB architecture, each **Host Controller** (HC) must implement a USB root hub.

> The **root hub** is a pseudo device that gets attached to the root of the USB device tree topology located at the rear of each HC.
>
> https://www.ibm.com/docs/en/aix/7.2?topic=interface-root-hub-emulation

2. ***Integrated Webcam_FHD*** - встроенная веб-камера ноутбука.
3. Bluetooth-адаптер ноутбука. Класс E0h определен для беспроводных контроллеров. Позволяет ноутбуку подключаться к различным Bluetooth-устройствам, например наушники или беспроволная мышь.

Также используя Vendor ID и Product ID, а также команду ***lsusb -tv***, обнаружил использование устройства:

- **Intel AX211 Bluetooth**
  (ID: 8087:0033; https://linux-hardware.org/?id=usb:8087-0033)
- **Sonix Technology Integrated Webcam_FHD**
  (ID: 3277:0004; Sonix Technology Integrated Webcam_FHD - Linux Hardware Database)

ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы я познакомился с библиотекой алгоритмов компьютерного зрения OpenCV а также освоил работу с встроенной камерой, изображениями, полученными с камеры, вывод обработанного изображения на экран и работу с окнами.

ПРИЛОЖЕНИЯ

Приложение 1: Исходный код программы *main.cpp*

```
#include <iostream>
#include <libusb.h>

void printDeviceInfo(libusb_device *dev) {
   libusb_device_descriptor desc;
   int r = libusb_get_device_descriptor(dev, &desc);
   if (r < 0) {
      std::cerr << "Ошибка: не удалось получить дескриптор устройства, код: " << r <<
std::endl;
      return;
   }

   // Выводим класс устройства, идентификатор производителя и идентификатор изделия
   std::cout << "Класс устройства: " << (int)desc.bDeviceClass << std::endl;
   std::cout << "Идентификатор производителя: " << (int)desc.idVendor << std::endl;
   std::cout << "Идентификатор устройства: " << std::hex << desc.idProduct << std::endl;

   libusb_device_handle *handle; // хендлер где будем хранить конфигурации
   r = libusb_open(dev, &handle);

   // Получаем и выводим серийный номер устройства
   if (r == 0) {
      unsigned char serialNumber[256];
      unsigned char Manufacturer[256];
      unsigned char Product[256];
      // получить дескриптор  устройства в виде строки символов
      if (libusb_get_string_descriptor_ascii(handle, desc.iSerialNumber, serialNumber,
sizeof(serialNumber)) > 0) {
         std::cout << "Серийный номер: " << serialNumber << std::endl;
      }

      if (libusb_get_string_descriptor_ascii(handle, desc.iManufacturer, Manufacturer,
sizeof(Manufacturer)) > 0) {
         std::cout << "Производитель: " << Manufacturer << std::endl;
      }

      if (libusb_get_string_descriptor_ascii(handle, desc.iProduct, Product, sizeof(Product)) > 0) {
         std::cout << "Изделие: " << Product << std::endl;
      }

      libusb_close(handle);
   } else {
```

```
        std::cerr << "Ошибка: не удалось открыть устройство." << std::endl;
    }

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

int main() {
    libusb_device **devs; // указатель на указатель на устройство, используется для
получения списка устройств
    libusb_context *ctx = NULL; // контекст сессии libusb
    int r; // для возвращаемых значений
    ssize_t cnt; // число найденных USB-устройств

    // Инициализируем libusb
    r = libusb_init(&ctx);
    if (r < 0) {
        std::cerr << "Ошибка: не удалось инициализировать libusb, код: " << r << std::endl;
        return 1;
    }

    // Получаем список подключенных USB-устройств
    cnt = libusb_get_device_list(ctx, &devs);
    if (cnt < 0) {
        std::cerr << "Ошибка: не удалось получить список устройств." << std::endl;
        libusb_exit(ctx);
        return 1;
    }

    std::cout << "Найдено USB-устройств: " << cnt << std::endl;
    std::cout << "==========================" << std::endl;

    for (ssize_t i = 0; i < cnt; i++) {
        printDeviceInfo(devs[i]);
    }

    // Освобождаем ресурсы
    libusb_free_device_list(devs, 1);
    libusb_exit(ctx);

    return 0;
}

// use sudo ./executable to run
```

Приложение 2: пример результата работы программы