The project's goal is to expose to how real kernel developers work - learning about application and theory on your own, finding appropriate examples or code bases to work with, and adapting existing libraries and tools to your needs. You will practice hands on development of USB device driver.

How the code works

When usb driver registerd to Linux USB subsystem it needs to give it some information about which devices the driver supports and which functions to call when a device supported by the driver is inserted or removed from the system. All of this information is passed to the USB subsystem in the usb_driver structure.

after the USB driver is registerd to the subsystem we plugged the device into the system with probe () function it created a node "/dev/os2Ex_dev {minor number}" to communicate between the user and the module.

static struct usb_driver myDevice_driver
the structure fields: the driver's name, ID table for auto-detecting the particular device and the 2 callback functions to be invoked by USB core during hot-plugging and hot-removal of the device.

Int usb_register(&myDevice_driver)
register the USB driver with the USB subsystem when entering the module to the kernel.

Int usb_deregister(&myDevice_driver)
deregister this driver with the USB subsystem this done in exit () function.

static int my_probe_function (struct usb_interface *interface, const struct usb_device_id *id)
Called when drivers found match in struct usb_device_id, Create a node in /dev/ with the name os2Ex_dev {minor number}.

static void my_disconnect_function (struct usb_interface *interface)
called when our device is unplugged, delete /dev/os2Ex_dev {minor number} node that created.

Moreover, as the file operations (write, read) are now provided, that is where exactly we need to do the data transfers to and from the USB device, so my_read and my_write using the usb_bulk_msg to do the transfers over bulk end points (out = 0x02, in = 0x81).

static ssize_t my_read (struct file *f, char __user *buf, size_t sz, loff_t *off)
called when the user transfer data from the USB device, we call the usb_bulk_msg function, giving it a buffer into which to place any data received from the device and a timeout value. If the timeout period expires without receiving any data from the device, the function will fail and return an error message.

static ssize_t my_write (struct file *f, const char __user *buf, size_t sz, loff_t *off)
called when the user transfer data to the USB device, this function copies the data from user space to kernel space and send the device_buffer to usb device via usb_bulk_msg ().

User Guide

How to compile the project:

- To compile **usb driver + char device driver** modules:

Run “make” command using terminal

```
user@ubuntu:~/OS_2/final_project$ make
make -C /lib/modules/5.4.0-58-generic/build M=/home/user/OS_2/final_project modu
les
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-58-generic'
  CC [M]  /home/user/OS_2/final_project/usb_driver.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/user/OS_2/final_project/usb_driver.mod.o
  LD [M]  /home/user/OS_2/final_project/usb_driver.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-58-generic'
```

```
user@ubuntu:~/usbproject/charDevice$ make
make -C /lib/modules/5.4.0-52-generic/build M=/home/user/usbproject/charDevice m
odules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-52-generic'
  CC [M]  /home/user/usbproject/charDevice/charDevice.o
  Building modules, stage 2.
  MODPOST 1 modules
```

- To compile the user program:

Run “user_program.c -o user_prog” using terminal

```
user@ubuntu:~/OS_2/final_project$ gcc user_program.c -o user_prog
user@ubuntu:~/OS_2/final_project$
```

How to run the project

- To run the **usb_driver** module:

Run “sudo insmod usb_driver.ko” to insert module to the kernel
You can see the module running the command “lsmod”

```
user@ubuntu:~/OS_2/final_project$ sudo insmod usb_driver.ko
[sudo] password for user:
user@ubuntu:~/OS_2/final_project$ lsmod
Module                  Size  Used by
usb_driver             16384  0
```

Run the command “demsg -w” Plug in the usb device to see the minor number

```
[28530.499205] Minor obtained: 3
```

As we see there is “os2EX_dev3 “created in /dev when the usb device is plugged in (the minor number is 3)

```
user@ubuntu:~$ cd /dev
user@ubuntu:/dev$ ls
autofs           hpet          mem
block            hugepages     mqueue
bsg              hwrng         net
btrfs-control    i2c-0         null
bus              initctl       nvram
cdrom            input         os2Ex_dev3
```

- To run the **char device driver** module:

Run “sudo insmod charDevice.ko” to insert module to the kernel
You can see the module running the command “lsmod”
this will create a new node in /dev also.

```
user@ubuntu:~/usbproject/charDevice$ sudo insmod charDevice.ko
user@ubuntu:~/usbproject/charDevice$
```

- To run the user program:

Run in terminal “sudo ./user_prog minor_number “any message”, this program works with “char device” and “usb driver” modules.

```
user@ubuntu:~/OS_2/final_project$ sudo ./user_prog 3 "hi there"
message from usb device: ereht ih
user@ubuntu:~/OS_2/final_project$ sudo ./user_prog 3 "who are you?"
message from usb device: OS2 course USB device
```

The program will show to the user an error message if one of the arguments is messing.

```
user@ubuntu:~/OS_2/final_project$ sudo ./user_prog 3
SYNTAX ERROR: PROGRAM MINOR_NUMBER "YOUR_MESSAGE"
user@ubuntu:~/OS_2/final_project$
```