UF UNIVERSITY of FLORIDA
Herbert Wertheim College of Engineering

COP4600

P4: Device Drivers

Overview

It's a great life when you're living the dream, working for the retro-gaming ScaleTech company. Soon, your company will be releasing the latest iteration in the classic GatorRaider game line, but this time, with a twist – it comes with a custom gaming device known as the ChompStick. Naturally, in typical game industry fashion, you don't actually have the hardware – it's still being finalized in the manufacturing division – so you're working with a virtual USB device that mimics its behavior. The actual hardware will surely bring a few surprises, but the team needs to get it as close as possible using what you currently have.

In this project, you will implement a userspace device driver that reads data from the virtual USB device and route the results to the standard Linux device system in Reptilian. Your device driver must allow the virtual device to be treated exactly like any other joystick device – to the point where it can be used to control software that uses typical joystick input. It's crunch time – the game is being released before the hardware (because ***of course it is***), and you only have 3 weeks to get the driver ready!

NOTE: Take Snapshots in VirtualBox! It's easy to cause crashes when working in the USB subsystem.

Structure

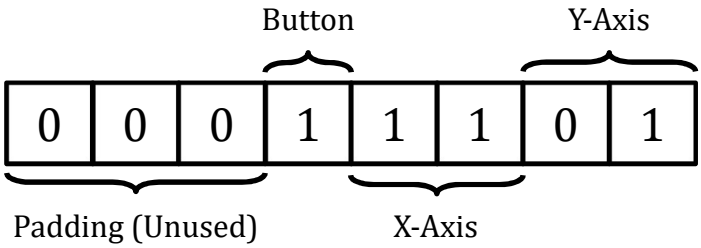
The project is broken into three main steps:

1) Connect to and read data from the device in the USB subsystem
2) Interpret the data (per the device specification) to determine the current device state
3) Route the data back into the Linux input system as a conventional joystick

While exact implementation may vary, the driver name and functionality must match the specification laid out in this document.

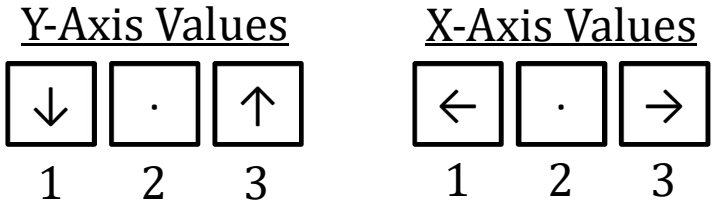
USB Device Specification

The ChompStick device, in USB parlance, is an ***interrupt-driven*** device with a single ***endpoint*** – that is, it sends data along a single communications channel whenever a request is sent to the device. Whenever the USB subsystem polls the ChompStick device, it sends a single byte of data, formatted as follows:



Byte Value	0x1D
X-Axis	3
Y-Axis	1
Button	1

When the button is pressed, the button bit will be set (1); otherwise, the bit will be unset (0). Each axis has a range from 1 to 3 – with 1 representing the lowest value:



ChompStick has a Vendor ID value of **`0x9A7A`** and a Product ID value of **`0xBA17`**.

Driver Specification

The userspace driver will be a stand-alone executable. As you are building a prototype driver, it is not necessary for it to run in the background; for testing purposes it will be run in a separate terminal. (The mainline kernel team will build on your work later.) The driver executable should be named **`chompdrv`** and should not require any parameters or other dependencies to successfully execute – as long as the USB device is present, it should route its state information to the joystick device.

When the driver is successfully implemented, the **`jstest`** tool (outlined below) should yield these values:

ChompStick Button	
JS Field	*Button 0*
If 0	*off*
If 1	*on*

ChompStick X-Axis	
JS Field	*Axis 0*
If 0	*(Invalid)*
If 1	*-32767*
If 2	*0*
If 3	*32767*

ChompStick Y-Axis	
JS Field	*Axis 1*
If 0	*(Invalid)*
If 1	*32767*
If 2	*0*
If 3	*-32767*

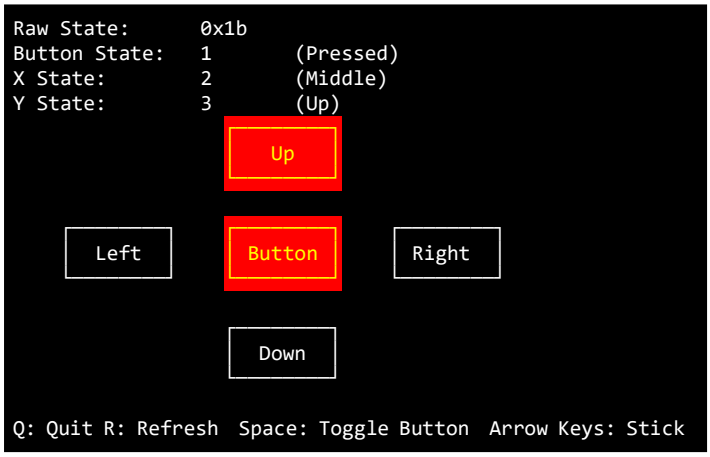
Running the Device, Driver, & Game

In order to use the virtual device, a few setup steps are necessary.

1) Install the new kernel configuration (in in the Project folder on Canvas) in **`/usr/rep/build/config`**
2) Copy the virtual device and utilities (in **`chompstick.tgz`**) to the virtual machine
3) Rebuild and reinstall the kernel and kernel modules
4) Update your local package index: **`$ sudo apt-get update && sudo apt-get upgrade`**
5) Install the USB and joystick tools: **`$ sudo apt-get install libusb-1.0-0-dev usbip joystick`**

Running the Virtual Device

Running the device requires two terminals: one to control the device, and another to collect data from it.



Terminal 1

Run the **chompapp** program. This creates the virtual USB device and displays a text-based controller for it.

Terminal 2

1) Connect the device to the USB subsystem:
   ```
   $ sudo usbip -a 127.0.0.1 1-1
   ```

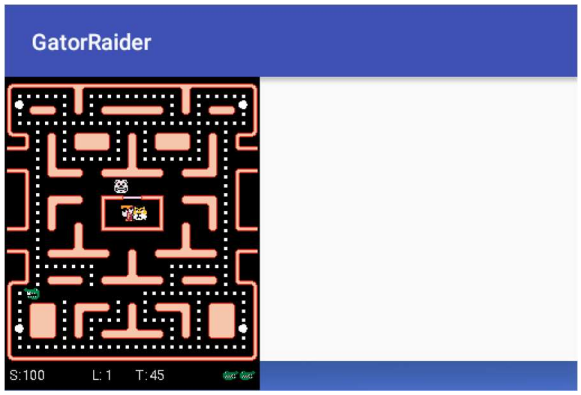
2) Run the **chompread** program to fetch and display the current state of the USB device.

Note that this is only showing the USB raw data – ***it isn't connecting the device as a conventional joystick!*** Once you finish writing the driver, it will bridge the gap between the text controller and the joystick system.

Running the GatorRaider Game

The GatorRaider game can be used for testing the device in the Android GUI. Follow these steps to install it:

1) Open "Settings" from the menu; navigate to "Security"
2) Turn on "Unknown sources", and click "OK"
3) Open Chrome in the Reptilian GUI
4) Go to **http://cise.ufl.edu/~jblanch/os/gatorraider.apk**.
5) Once downloaded, click "open", then "Install".



You can run the app any time; go to "GatorRaider" from the main menu. It can be controlled via arrow keys or a joystick. You can plug in an XBox controller (or many other joysticks) and connect it to VirtualBox via the "Devices" → "USB" drop-down menu (along the top of the VM). Once your joystick driver is working, you'll be able to control the character via the console app.

Testing Your Driver

To test your driver, you should run the **chompapp** program in a terminal as described earlier. In a second terminal, you will need to connect it to the USB subsystem, then run your driver:

```
$ sudo usbip -a 127.0.0.1 1-1
$ sudo ./chompdrv
```

Run only once, after **chompapp**!

Using Yet-Another Terminal

To test from the command line, open ***yet another*** terminal window on Reptilian and run **jstest**:

```
$ jstest /dev/input/js0
```

If your driver is successfully connected, you'll see information about its buttons and axes. Note that if you've connected an external controller, **js0** may be that external device. Be careful!

Using GatorRaider

The GatorRaider game is all set to use joystick controls. As such, once your driver is running, if it is properly functioning, it should automatically pick up input from this driver and use it to move the gator character!

Submissions

You will submit the following at the end of this project:

- Report on Canvas
- Screencast on Canvas
- Compressed tar archive (**chompdrv.tar.gz**) for **chompdrv** userspace driver on Canvas

Report

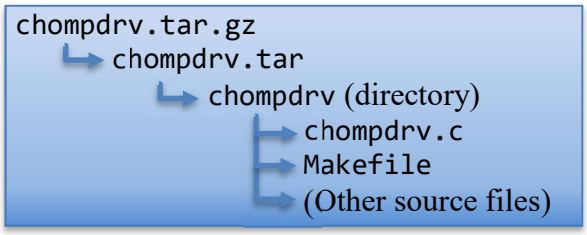
Your report will explain how you implemented the driver, including how you connected to the USB subsystem as well as the joystick input subsystem. It must include an explanation of the structures used and in what ways. It will include description of how testing was performed along with any known bugs. The report may be in Portable Document Format (pdf) or plain-text (txt) and should be no more than a page. It should cover all relevant aspects of the project and be organized and formatted professionally – ***this is not a memo!***

Screencast

In addition to the written text report, you should submit a screencast (with audio) walking through the driver you wrote to connect the raw USB device to the joystick input system (~5 minutes).

Compressed Archive (classification.tar.gz)

Your compressed tar file should have the following directory/file structure:



To build the driver, we will execute these commands:

```
$ tar zxvf chompdrv.tar.gz
$ cd chompdrv
$ make
$ cd ..
```

To run the driver, we will execute this command:

```
$ chompdrv/chompdrv
```

Please test your driver build and linking before submission! If your driver does not compile it will result in **zero credit** (0, none, goose-egg) for the code portion of the project.

Helpful Links

You may find the following resources helpful when reading about how write the userspace driver:

https://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/
https://www.kernel.org/doc/html/v4.12/input/uinput.html
https://stackoverflow.com/questions/16032982/getting-live-info-from-dev-input