**Linux Keyboard Driver**

**Alanah Bell | Wesley Walker**

**COSC 439 - Dr. Ramesh Karne**

***Table of Contents***

1. ***Objective***
2. ***Introduction to Keyboard Driver***
3. ***Installation and Running***
4. ***Different Keyboard Effects***
5. ***Analysis and Customization***
6. ***Custom Effects***
7. ***Obstacles and Recap***
8. ***References***

***Objective***

*For our project, we decided to focus on keyboard drivers. Specifically we researched and experimented with the open source keyboard driver for Razer products. Razer provides a Github repository that can be found at* [*https://github.com/openrazer/openrazer*](https://github.com/openrazer/openrazer)*. We are using a Razer Cynosa Chroma Pro. Our plan is to research and experiment with the Razer driver enough to create our own keyboard effect with the lighting and provide analysis on our findings.*

***Introduction to Keyboard Drivers***

*Keyboard drivers act as the interface between the I/O device keyboard and the actual computer. The keyboard itself sends a binary stream of data to the computer containing keycodes. The raw keycodes are not readable by the computer. The driver formats these keycodes so that the OS can understand. The OS and keyboard driver act together as middleware for the computer and keyboard.*

***What is a driver?***

*A driver is a set of files that tell the hardware how to function. It lets the operating system and a device communicate with each other. Some examples of drivers are disk drives, video cards, printers, and in our case, keyboards. The driver has a special set of commands for a specific device.*

***What is a daemon?***

*A daemon is a long-running program that executes as a background process in Linux/Unix operating systems. It performs tasks when required and answers requests for services.*

***How do keyboards interact with the OS?***

*When you press a button on a keyboard, its correlating scan code is sent to the CPU as a binary number. The operating system is constantly checking for new scan codes and converts that scan code into ASCII. The operating system then creates an event for the button pressed, which is sent back to the application currently running. Lastly, the application processes the associated task accordingly.*

***Installation***

***Installing Open Source Razer Repository:***

*Due to our project being done in Linux, Ubuntu was installed onto our computers. When starting the process of accessing the repository and joining the plugdev group in the terminal, an error was received. It was discovered that secure boot had to be disabled in the PC BIOS menu. We were able to proceed from there, with no other issues regarding permissions or headers as we continued preparing our machine for the Razer keyboard driver.*

*To correctly download the open source Razer repository on Ubuntu, we ran the following lines in the terminal:*

sudo add-apt-repository ppa:openrazer/stable

sudo apt update

sudo apt install openrazer-meta

*The first line downloads the PPA, or Personal Package Archive, which allows Ubuntu users to install the repository easily. The next two lines are a way to update and install the newest version of the software.*

***Building the program:***

*Building the program configures and compiles the source code. To do this, access the repository folder in the directory and run “make”.*

*Once this is complete, the software has been built and is ready for execution. The driver folder of the repository will now have .o files associated with each original file. Files with the .ko extension are also available now. Files with the .o extension have compiled object code, while those with the .ko extension represent LKMs, or Loadable Kernel Modules. After learning more about LKM, we were able to determine that we needed a KO module for our keyboard driver to work. More specifically, this is the module that will be loaded for our driver to be functional.*

***Insert module into kernel:***

*Before adding the razer keyboard module, the keyboard will automatically use the default linux driver. Our goal is to load our new module and bind it to the razer keyboard.*

*First, you must find the correct target file. The file specifically for a keyboard is named razerkbd.ko.*

*To insert it as a module do the following:*

*sudo insmod /usr/src/openrazer-driver-3.1.0/driver/razerkbd.ko*

*To remove a module:*

*sudo modprobe -r <module name>*

*To check if module has been removed or added:*

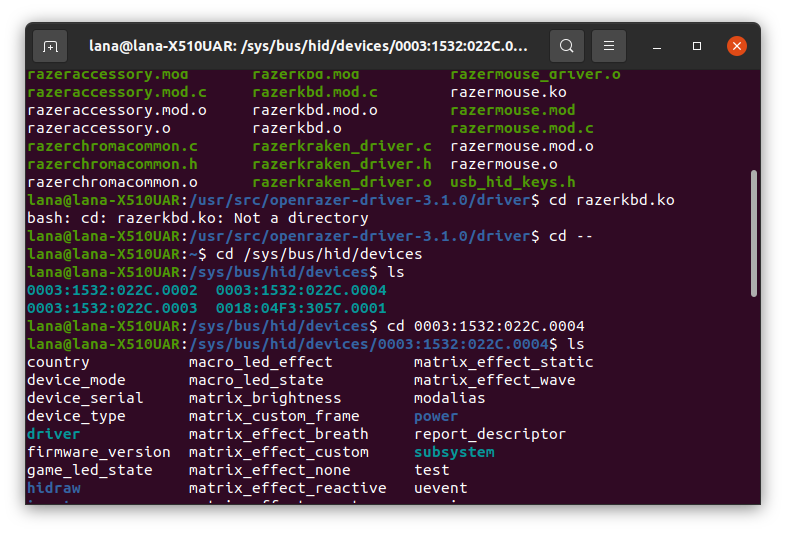
*sudo lsmod | grep razerkbd*

***Binding driver to keyboard:***

*The keyboard now needs to be bound to the correct driver. First, locate the keyboard’s HID address:*

*cd /sys/bus/hid/devices/*

*ls*

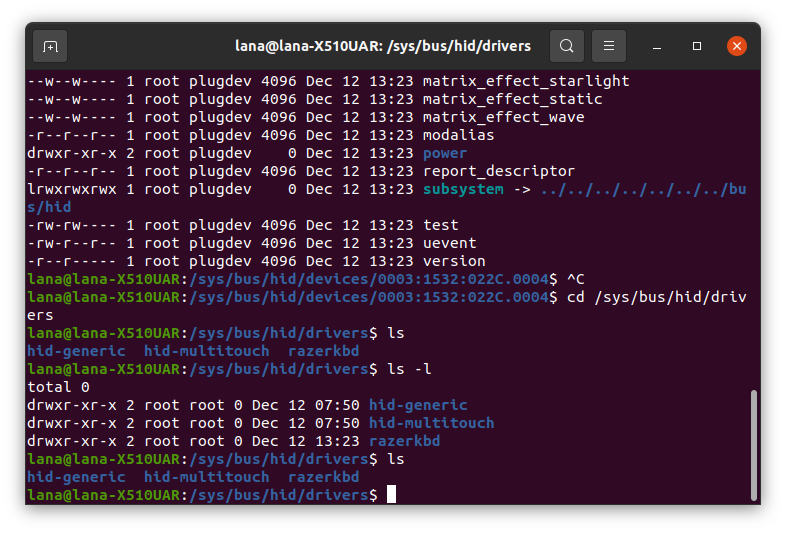
**

*Our specific model of keyboard has an ID number of 1532:022C. Out of the devices with this ID in the center, the one with the highest ending number is the correct address.*

*Now, check the driver folder to see what the devices are bound to.*

*cd /sys/bus/hid/drivers/*

*ls*

**

*To unbind:*

*In /sys/bus/hid/drivers/hid-generic/0003:1532:022C.0004 pathway,*

*echo -n "0003:1532:022C.0004 > /sys/bus/hid/drivers/razerkbd/unbind*

*To bind:*

*In /sys/bus/hid/drivers/razerkbd/0003:1532:022C.0004 pathway,*

*echo -n "0003:1532:022C.0004 " > /sys/bus/hid/drivers/razerkbd/bind*

*At this point, the razer keyboard driver module has been loaded to the kernel and bound to the device. The keyboard is now ready for use.*

***Different Keyboard Effects***

*Once you have the Razer Driver installed you are able to start testing out different keyboard effects. You are given access to a variety of different effects. To initiate the effect, you must type out “echo -n” followed by the command associated with it.*

***Analysis and Customization - Example 1 [Matrix Effect Breath]***

***$ echo -n "1" > matrix\_effect\_breath*** *[example a]*

***$ echo -n -e "\xFF\x00\x00" > matrix\_effect\_breath*** *[example b]*

***$ echo -n -e "\xFF\x00\x00\x00\x00\xFF" > matrix\_effect\_breath*** *[example c]*

*You can notice how there is an argument written within the quotations. The argument passed is contingent not only on what effect you are calling, but also the variation of the effect. The three cases above show very different arguments passed for the same effect. You can also notice how there is an* ***-e*** *included before the argument for the second two examples. This inclusion indicates you are specifying the color(s) you want displayed through the effect. [Example b] gives a 3-byte argument specifying the single color to be interpreted as RGB. [Example c] gives a 6-byte argument to signify dual color mode. The first 3 bytes for RGB color 1, the second 3 bytes for RGB color 2. If you give a 1-byte argument, like in [example a], random mode will be activated.*

***Analysis and Customization - Example 2 [Matrix Effect Starlight]***

***$ echo -n -e "\x02" > matrix\_effect\_starlight***

***$ echo -n -e "\x02\xFF\x00\x00" > matrix\_effect\_starlight***

***$ echo -n -e "\x02\xFF\x00\x00\x00\x00\xFF" > matrix\_effect\_starlight***

*Similar to the last example, this effect takes arguments and includes a default setting. The default setting you exclude the byte(s) for declaring RGB color(s) and you are given a random one. You may think that the default command would exclude* ***-e*** *based off of what is seen in* ***Example 1****. However, you must include* ***-e*** *when giving an argument other than option selection such as in* ***Example 1*** *[example a]. It is clear to see when an RGB argument is being given, but other arguments depend on the effect.* ***Example 2*** *takes a 1-byte parameter before the RGB argument(s) declaring the decay speed of the color within the Starlight effect.*

***Custom Effects - Overview***

*We have shown how through the command line, you can call different effects with custom parameters. Here you can give arguments that dictate colour, speed, brightness, etc. Creating custom effects is not limited to just changing parameters in the command line call. You can also go into the driver code written in C.*

***Custom Effects - Configuration***

***The Razer Report***

*To learn how the device controller can receive information and perform an effect on the physical keyboard, we began our search in razerkbd\_driver.c. This includes functions and effects only capable of being performed by the keyboard exclusively. (Razer has other products, like mice and headphones, that share similar files with the keyboard in the driver folder.) When looking through different functions in razerkbd\_driver.c, we noticed variables and functions that originated from other files. With this information and the list of files included at the top of the file, we were able to determine what other files would be important for our keyboard driver. Overall, there were 3 files we mainly worked with: razerkbd\_driver.c, razercommon.c, and razerchromacommon.c.*

*After this, our goal was to follow the code to see how it communicates information to the keyboard and what information must be included for each type of effect. After tracing the code of a few functions, we noticed that no matter the type of effect, a “razer\_report” was being returned. It is a data structure, found in razercommon.c, that holds all data necessary to perform an effect. Not all of the variables within the structure are necessary to perform a function as it will depend on the device and specific function. However, a command\_class, command\_id, and data\_size (all unsigned chars) will always be required and are needed to initialize a report.*

*Get initialised razer report:*

*struct razer\_report get\_razer\_report(unsigned char command\_class, unsigned char command\_id, unsigned char data\_size)*

*{*

*struct razer\_report new\_report = {0};*

*memset(&new\_report, 0, sizeof(struct razer\_report));*

*new\_report.status = 0x00;*

*new\_report.transaction\_id.id = 0xFF;*

*new\_report.remaining\_packets = 0x00;*

*new\_report.protocol\_type = 0x00;*

*new\_report.command\_class = command\_class;*

*new\_report.command\_id.id = command\_id;*

*new\_report.data\_size = data\_size;*

*return new\_report;*

*}*

*Get empty razer report:*

*struct razer\_report get\_empty\_razer\_report(void)*

*{*

*struct razer\_report new\_report = {0};*

*memset(&new\_report, 0, sizeof(struct razer\_report));*

*return new\_report;*

*}*

*As we began researching what each individual data point represents, we saw how important the arguments[] variable is. It has an array with 80 elements, which will allow a function to have multiple arguments, which is often necessary. An argument could represent an LED ID, RGB based colors, an effect ID, storage of space, or starting and stopping points of columns on keyboard layout to name some examples. It became clear that this array allows high flexibility for keyboard functionality.*

*Notice the memset() function in the razer report code. It works to save the contents of the razer report into memory. The keyboard will perform the effects given from the razer report until another function that changes at least one of the elements is called. This includes times when the keyboard is unplugged or plugged into a different machine.*

***Custom Effects - Editing the Code***

*Once you are set up you can start editing the code. The best way to experiment is just to mess around and see what works. For us, we started with simple changes.*

***Setting Ranges***

*In razerchromacommon.c:*

*struct razer\_report razer\_chroma\_extended\_matrix\_effect\_starlight\_random(unsigned char variable\_storage, unsigned char led\_id, unsigned char speed)*

*{*

*struct razer\_report report = razer\_chroma\_extended\_matrix\_effect\_base(0x06, variable\_storage, led\_id, 0x07);*

*speed = clamp\_u8(speed, 0x01, 0x03);*

*report.arguments[4] = speed;*

*return report;*

*}*

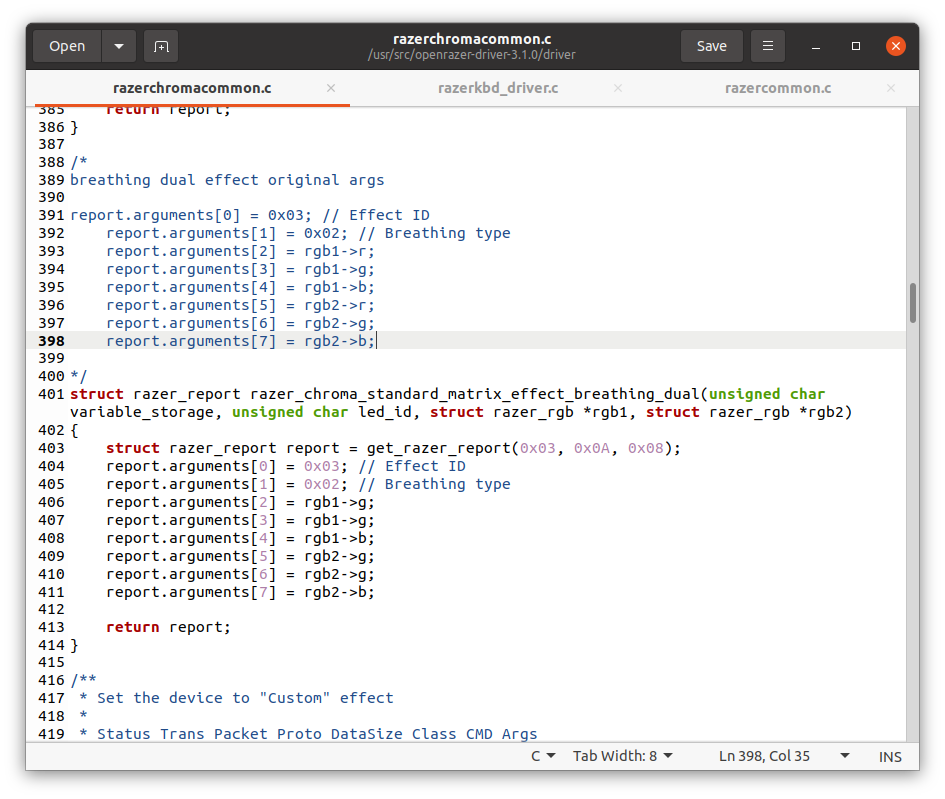
*The code above uses the clamp() method for speed. After investigating, we learned that this method is used to set ranges. The first parameter defines the value, the second is the lower bound, and the third is the upper. In this case, speed can be set from 1-3. When testing this, we discovered 0x01 represents the fastest speed and 0x03 is the slowest speed.*

*We decided to modify the starlight effect above by limiting the range of speeds it could display.*

*speed = clamp\_u8(speed, 0x01, 0x02);*

*Now, it can only operate at the fastest to second fastest speed. If you try to set it to 3 in the terminal, nothing will happen.*

***Color Capabilities***

**

*Above you can see the original code commented out and then our modification below. Here we alter the parameters being passed to each rgb assignment. In this case we take away the red from the argument in both colour assignments. So, now if you were to call Blue and Green as your colours in matrix\_effect\_breathing\_dual you will get Blue and Yellow. This is because we took away the inclusion of Red which Green needs to become Green.*

*Many effects had the option to use multiple colors. After comparing code of these functions, we noticed that the addition or removal of razer report arguments makes this possible.*

*In razerchromacommon.c:*

*struct razer\_report razer\_chroma\_extended\_matrix\_effect\_reactive(unsigned char variable\_storage, unsigned char led\_id, unsigned char speed, struct razer\_rgb \*rgb)*

*{*

*struct razer\_report report = razer\_chroma\_extended\_matrix\_effect\_base(0x09, variable\_storage, led\_id, 0x05);*

*speed = clamp\_u8(speed, 0x01, 0x04);*

*report.arguments[4] = speed;*

*report.arguments[5] = 0x01;*

*report.arguments[6] = rgb->r;*

*report.arguments[7] = rgb->g;*

*report.arguments[8] = rgb->b;*

*return report;*

*}*

*Take a look at the reactive function above. It’s razer report arguments 6, 7, and 8 are all RBG arguments. Each one represents either red, green, or blue, which together make up the entire color spectrum. Together, they represent a structure that will store a certain color, given in the terminal. For dual effects, add 3 more RGB arguments and a new parameter like so:*

*struct razer\_report razer\_chroma\_extended\_matrix\_effect\_reactive(unsigned char variable\_storage, unsigned char led\_id, unsigned char speed, struct razer\_rgb \*rgb1, struct razer\_rgb \*rgb2)*

*report.arguments[4] = speed;*

*report.arguments[5] = 0x01;*

*report.arguments[6] = rgb->r;*

*report.arguments[7] = rgb->g;*

*report.arguments[8] = rgb->b;*

*report.arguments[9] = rgb->r;*

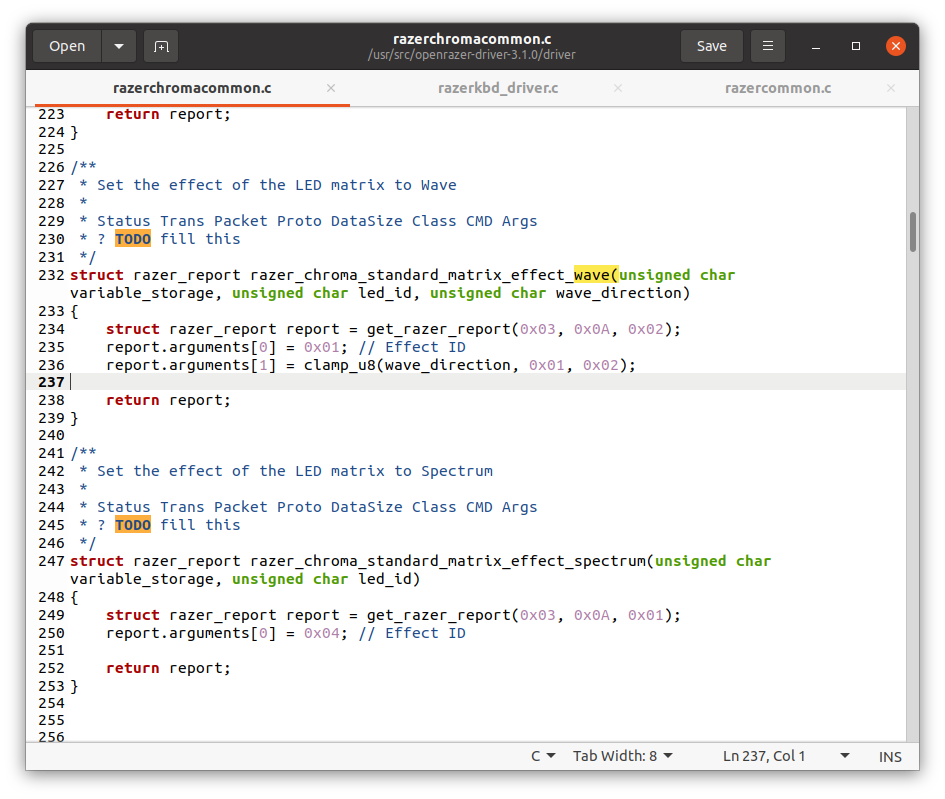
*report.arguments[10] = rgb->g;*

*report.arguments[11] = rgb->b;*

*With two RGB structures, the keyboard will circulate through these bytes being passed to it. It will start with arguments 6-8, the first structure, and end with arguments 9-11, the second structure. After this, it will return back to the first one and repeat.*

*Going further into ideas for editing the code you can think about combining effects. One example is having one effect start, end, another effect is called, ends and then the initial effect begins again in a recursive manner. Similar to this you could try this with two different effect types of the same effect (of the same effect ID).*

*For example:*

**

*In this case you see the wave effect can be given a direction argument. It either can wave Left -> Right or Right-> Left. However, what if you could make an effect that combines the two and you can repeatedly call one after the other?*

***Obstacles and Recap***

*One of the biggest obstacles was successfully loading the razer keyboard driver module to the linux kernel. We made the mistake of cloning the repository twice, using different methods, so it became difficult to know which one the module was using. After guaranteeing our modified driver code was the only code available, it was easier to build and produce its module. Most issues were resolved using sudo and chmod 777. This included setting permissions for the files within the driver.*

*Overall, we discovered that Linux is a flexible system, allowing custom modules to be loaded to its kernel easily. We learned how keyboard drivers read information and how effect functions and the razer report can organize this data in the format and order required. Finally, we explored how different data points are represented as an argument, and the extent to which we could alter these arguments.*

***References***

*Driver Source Code:*

[*https://github.com/openrazer/openrazer*](https://github.com/openrazer/openrazer)

[*https://openrazer.github.io/#project*](https://openrazer.github.io/#project)

[*http://www.infotinks.com/driver-binding-and-unbinding-disappearing-and-appearing-drives/*](http://www.infotinks.com/driver-binding-and-unbinding-disappearing-and-appearing-drives/)

[*https://docs.fedoraproject.org/en-US/fedora/rawhide/system-administrators-guide/kernel-module-driver-configuration/Working\_with\_Kernel\_Modules/*](https://docs.fedoraproject.org/en-US/fedora/rawhide/system-administrators-guide/kernel-module-driver-configuration/Working_with_Kernel_Modules/)

[*https://www.kernel.org/doc/html/v4.16/driver-api/basics.html*](https://www.kernel.org/doc/html/v4.16/driver-api/basics.html)

[*https://www.oreilly.com/library/view/linux-device-drivers/0596005903/ch02.html*](https://www.oreilly.com/library/view/linux-device-drivers/0596005903/ch02.html)