Writeup 4

CS 444
November 26, 2018

Anton Synytsia, Eytan Brodsky, David Jansen

Contents

1 Introduction

The following writeup is based on Writeup 3 but with additional information regarding the character device, option to control speed, and option to set “one-off” or “repeat” modes.

In the following section, we answer specific questions associated with the assignment and provide initial guidance for the TA to evaluate our work.

1) We believe that the main point of this assignment was to get familiar with manipulating the Linux kernel. This assignment also forced us to get comfortable working with and manipulating drivers for the Linux operating system that directly interact with hardware. Specifically it was focused on getting us to work with the Linux LED driver, which interacts with the LED's on a Raspberry Pi. This assignment allowed us to manipulate a driver and see the results of our work being output to hardware in real time. This way we could easily test that our work was correct and our driver was functioning properly.
2) Our team approached the problem by first doing research on Morse Code. Before writing any code for the LED driver we began by writing out a phrase and it's corresponding Morse Code. Once we had a good understanding of how Morse Code worked we began diving in to our actual implementation. We considered doing the easiest approach of just outputting a single static phrase, we quickly changed our mind because we wanted to prepare for the next assignment. Instead we designed our driver to work with dynamically inputted phrases. We did this by creating a function that would convert each letter in the English language to it’s corresponding Morse Code. Once a letter was converted it would be processed by a function that would display the code on the LED board of our Raspberry Pi. This process would be repeated for each letter/word in the inputted phrase to the function until it is completed. Once the phrase is done being outputted to the board it will restart from the beginning of the phrase ,after a delay, and start the initial process all over again. This will continue infinitely until the process is stopped.
3) To ensure our Morse code LED trigger is correct, we made our trigger print out, via `printk`, the Morse code version of the sentence it is signalling. We also compared the light signaling pattern to an online Morse code emulator for consistency. Our Morse code LED trigger signals `"Linux operating systems"`. If the same string is used in an online Morse code emulator, the signalling pattern would be the same. The longevity of the message may be different but the pattern is identical.
4) For this assignment, we learned a couple of things, described below:
   a) There is a thing called, `jiffies`, which all the delays must be converted to when passed to the timer. According to Linux MAN page, a jiffy defines the inverse of an update frequency of the kernel. The update frequency of a kernel varies with hardware platform and kernel versions. Converting delays to jiffies allows for consistent timing under different Linux kernels and platforms. This is one of the things we learned.
   b) Another thing we learned is configuring `Kconfig` file. When adding a configuration entry for our Morse code LED trigger, we also had to insert `default y`, so that when a `.config` is generated, the Morse code is enabled without us having to modify the generated `.config` file afterwards.
5) To evaluate and grade our work, the TA can follow the sections below, in chronological order, and replicate our Morse code LED trigger for their Raspberry Pi. If the TA already have the Linux Kernel installed, Raspbian Light setup, and the serial console operating, they can get right into section 4.5 for testing our Morse Code LED trigger on their own Raspberry Pi. If the TA is experienced, they can review each section for legitimacy without replicating and contact

Group 1 for a demo of the Morse code LED trigger.

2 Required Technologies

The following technologies are required to perform this lab:

1) Raspberry Pi, preferably model 3B+,
2) Micro SD card and card reader, with at least 4GB space.
3) 3.3V TTL UART to USB converter for establishing serial console.
4) Power adapter or charger for Raspberry Pi.
5) Access to `OS2` server.

3 Setup

The following sections describe how to set up Raspbian Stretch Lite on Raspberry Pi. The first section describes how to setup Raspbian on SD card, the second section focuses on testing Raspbian with serial console, the third section describes how to build Raspberry Pi Linux kernel on `OS2`, and the fourth section describes how to setup a built Linux kernel image on SD card.

3.1 Downloading and Setting Up Raspbian

Refer to the following steps for setting up Raspbian Stretch Lite on SD card:

1) Download and extract `2018-10-09-raspbian-stretch-lite.zip` from https://www.raspberrypi.org/downloads/raspbian/.
2) Download and install Etcher from https://www.balena.io/etcher/.
3) Mount the micro SD card to your laptop, via the SD card reader.
4) Start Etcher and do the following:
   a) Set image to `2018-10-09-raspbian-stretch-lite.img` (or alike).
   b) Set drive to SD card.
   c) Click textitFlash!
5) Once the setup is complete, navigate to the SD card drive and use text editor to append the following to `config.txt`:

```
kernel=kernel8.img
enable_uart=1
```

3.2 Running Raspbian with Serial Console

The following steps, heavily based on Adafruit guide, describe how to initiate a Raspbian serial console session:

1) Install Prolific Chipset and SiLabs CP210X drivers for the TTL serial cable [1].
2) Connect black, white, and green wires to the outer pins 3, 4, and 5 respectively [2].
3) Leave red wire unpinned, as the a separate power adapter is used instead [2]. It is important that only one power source is used as the board can get damaged [2].
4) Insert the micro SD card into Raspberry Pi.

5) Insert the TTL serial cable USB into your laptop.
6) Start Putty and do the following:
   a) Set *Connection type* to serial mode.
   b) Set *Serial line* to `COM6`; COM6 here refers to the port of our TTL serial cable. To determine the port of your TTL serial cable, on Windows platform, access *Device Manager* and check for the available ports; for other platforms, refer to Adafruit guide [2].
   c) Set *Speed* to 115200 [2].
   d) Click *Open*.
7) Connect the power adapter to Raspberry Pi. It is important that this step is performed after initiating the serial console session.
8) (Optional) Within the serial console, press *RETURN* key to activate communications [2].
9) After loading, use `pi` as user name and `raspberry` as password.

3.3 Raspberry Pi Linux Kernel

3.3.1 Setting Up

Perform the following steps for downloading and setting up version 4.14.y Raspberry Pi Linux kernel on `OS2`:

```
cd /scratch/fall2018/group1
git clone git@github.com:raspberrypi/linux.git
cd linux
git checkout tags/raspberrypi-kernel_1.20180417-1 # checkout v4.14.y
```

3.3.2 Compiling

Execute the following set of commands to build Raspberry Pi Linux Kernel on `OS2`:

```
cd linux
KERNEL=kernel8
make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- bcmrpi3_defconfig
make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- all
```

Once built, refer to the next section for setting up the built image on SD card.

3.3.3 Uploading to SD Card

Refer to the following steps for setting up `kernel8.img` on SD card:

1) Download `Image` from `arch/arm64/boot/Image` to your local file system, either using WinCP or another file transfer protocol. An easy way to do it is to first copy to base path and then download with WinCP to your computer:

   ```
   cd linux
   cp arch/arm64/boot/Image ~/kernel8.img
   ```

2) Rename `Image` to `kernel8.img`.
3) Mount SD card to your laptop.
4) Copy `kernel8.img` to SD card, so that it is located at the same path as `kernel7.img`.

4 Morse Code LED Trigger

The following sections describe our solution to Raspberry Pi Morse code LED trigger, as well as, instructions for compiling, setting up, and running the blinker.

4.1 Solution

1) We began by creating an array of each letter in the alphabet represented by it's Morse Code value.
2) We then created a function that would convert ASCII characters to their Morse Code value (This was done in order to make assignment 3 easier). This was done by subtracting each ASCII character by 0x41, this would make it so the new value would point to the index of the Morse code value of the ASCII character. For the space character we subtracted by 0x61.
3) For displaying the Morse Code on the LED of the Raspberry Pi we used a struct, this struct is shown/explained below.

```
struct morse_trig_data {
 char* message;
 unsigned int indexL; // Letter index
 unsigned int indexM; // Letter part index
 unsigned int delayM; // This indicates whether to wait  or proceed to the next part of the letter.
 unsigned int invert;
 struct timer_list timer;
};
```

4) We also used a few global variables in order to set both the length of when the LED is turned on and when it is turned off while transmitting the Morse Code. These variables are shown/explained below.

```
const int DOT_LENGTH = 1; //Time LED will light up for a dot.
const int DASH_LENGTH = 3; //Time LED will light up for a dash.
const int MESSAGE_DELAY = 20; //Delay in between each retransmission of the message.
const int WORD_DELAY = 7; //Delay in between each word.
const int LETTER_DELAY = 3; //Delay in between each letter.
const int PART_DELAY = 1; //Delay in between each dot or dash for a letter.
const int DELAY_MAGNITUDE = 50;
```

5) Finally we get to our function that converts the Morse code over to the LED on the Raspberry Pi. The function starts by iterating through a word or phrase input to the function. First it checks if the word/phrase we are processing is already completed by checking for a null character, if it has it waits 20 units of time. Next the function will check if the character we are checking is a space, if it is, it will wait 7 units of time. If all of these tests pass the function knows it is processing a letter. If the function is processing a letter it will use the conversion function to convert the letter to it's Morse Code equivalent. Once the conversion is complete it will check to see if every part of the code is valid. If the code is valid it will check to see if the part of code it is processing is a dot or a dash. Dots will light up the LED for one unit of time, then the LED will be delayed/shut off for one unit of time. Dashes will light up the LED for three units of time, then the LED will be delayed/shut off for one unit of time. During this process the function also checks to see if the word/letter is complete yet, if the word is complete the LED will be delayed/shut off for a certain period

of time. The delay between letters is 3 units of time, the delay between words is 7 units of time. This same process will be repeated for every letter/word left in the inputted phrase.

4.2 Speed Control

To control Morse code speed, we utilize a device file and setup the show and store attributes for the device file. The device file is created with `device_create_file` function whenever the Morse trigger is activated and destroyed with the `device_remove_file` whenever the Morse trigger is deactivated. We use the `DEVICE_ATTR` to register the `led_speed_show` and `led_speed_store` attributes.

A buffer received within the `led_speed_store` callback function is converted to an integer, clamped between 1 and 10000, and saved to our `morse_trig_data` structure instance for use within the `led_morse_function` timer function:

```
mod_timer(&morse_data->timer, jiffies + msecs_to_jiffies(delay * morse_data->speed));
```

`led_speed_show`

Section 4.7 describes how to control speed when the Morse code trigger is ran from Raspberry Pi.

The show message just prints out the space

4.3 Repeat Control

For the repeat control we used a device file and utilized the show and store attributes of the device file in order show and change if the message will repeat infinitely or it will only display once. The device file is created with `device_create_file` function whenever the Morse trigger is activated and destroyed with the `device_remove_file` whenever the Morse trigger is deactivated. We use the `DEVICE_ATTR` to register the `led_repeat_show` and `led_repeat_store` attributes.

A buffer is received from the device file and using the callback function `led_repeat_store` the buffer containing the Boolean for the repeat functionality is converted to an integer. For the repeat control, it is restricted to being 1 or 0, if the value is a 1 that means it will repeat the message infinitely, if the value is a 0 the message will only be repeated a single time.

Section 4.7 describes how to control repeat mode when the Morse code trigger is ran from Raspberry Pi.

4.4 Message Control

For the message control we used a device file and utilized the show and store attributes of the device file in order to show and change the message that is currently being displayed on the raspberry pi.The device file is created with `device_create_file` function whenever the Morse trigger is activated and destroyed with the `device_remove_file` whenever the Morse trigger is deactivated. We use the `DEVICE_ATTR` to register the `led_message_show` and `led_message_store` attributes.

A buffer is received from the device file and using the callback function `led_message_store` the buffer containing the string is not converted because it is already in the correct variable type. For message control all ASCII characters are accepted, except the NULL terminator and new line characters, this is because they will end the message. Even though everything is accepted only digits and letters will be actually converted to Morse code, all other characters will be treated

as the space character, they will also output an error to the console log. When a new message is set it will terminate the previous message after it has finished displaying the Morse code for the last character it was transmitting. The new message will then instantly start displaying on the raspberry pi.

Section 4.7 describes how to assign message when the Morse code trigger is ran from Raspberry Pi.

4.5 Compiling

Perform the following steps to compile Raspberry Pi with our Morse code LED trigger:

1) Provided that Raspberry Pi Linux Kernel is cloned and checked out to the correct version at your local space, on `OS2`, copy `linux` folder, shipped with this repository, to your `linux` folder. This will overwrite and add the following files to `linux/drivers/leds/trigger/`:

   ledtrig-morse.c Our Morse code LED trigger.

   Kconfig Configures our Morse code LED trigger.

   Makefile Registers our Morse code LED trigger.

2) Executed the following set of commands to rebuild the kernel:

```
cd linux
make clean
KERNEL=kernel8
make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- bcmrpi3_defconfig
make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- all
```

4.6 Setting Up

Upload the new, built kernel image to SD card, described in section 3.3.3. You may have to delete the original `kernel8.img` from the SD card first though.

4.7 Running

Start a new serial console session, described in section 3.2. Then run the following set of commands to activate and control the Morse code LED trigger:

```
# After logging in, switch mode
sudo su
# optional: turn off lights
echo none > /sys/class/leds/led0/trigger
echo none > /sys/class/leds/led1/trigger
# launch
echo morse > /sys/class/leds/led1/trigger
# getting speed
cat /sys/class/leds/led1/speed
# setting speed (clamped between 1 - 1000)
```

```
echo 10 > /sys/class/leds/led1/speed
# getting message
cat /sys/class/leds/led1/message
# setting message
echo "Hello world" > /sys/class/leds/led1/message
# getting mode
cat /sys/class/leds/led1/mode
# setting "one-off" mode
echo 0 > /sys/class/leds/led1/mode
# setting "repeat" mode
echo 1 > /sys/class/leds/led1/mode
```

The message can contain letters `A-Z`, either uppercase or lowercase, digits `0-9`, and spaces. All unrecognized characters are replaced with a space delay and are printed out to console.

5 Version Control

acronym	meaning
V	`version`
tag	`git tag`
MF	Number of `modified files`.
AL	Number of `added lines`.
DL	Number of `deleted lines`.

V	tag	date	commit message	MF	AL	DL
1		2018-10-08	Initial commit	1	1	0
2		2018-10-08	Create .gitignore	1	6	0
3		2018-10-08	Update state	1	0	1
4		2018-10-08	Setup readme	1	5	0
5		2018-10-08	Update README.md	1	2	0
6		2018-10-08	Uploaded concurrency	2	225	0
7		2018-10-08	changed permissions	4	0	0
8		2018-10-09	Setup writeup TeX	2	126	0
9		2018-10-09	Added pygments	1	98	0
10		2018-10-09	included pygements to preamble	1	8	6
11		2018-10-09	First sucessful compile of writeup1	6	2008	2
12		2018-10-09	Updated gitignore	5	3	2005
13		2018-10-09	Setup Writeup	1	10	20
14		2018-10-09	Added git attributes to enforce EOL	1	7	0

Please continue at the next page

V	tag	date	commit message	MF	AL	DL
15		2018-10-09	EOL deal	2	1	2
16		2018-10-09	Trying to fix TEX compiling	2	79	1
17		2018-10-09	Added sections to writeup	3	8	81
18		2018-10-09	Added the writeup for Command Logs and for the Qemu flags	11	97	10
19		2018-10-10	Work on concurrency writeup	8	30	56
20		2018-10-10	More work on Concurrency writeup	2	12	7
21		2018-10-10	Made Concurrency use circular queue	1	41	34
22		2018-10-10	Fixed concurrency print type	1	9	6
23		2018-10-10	Renamed Assignment1 folder to Concurency and finished my writeup	7	251	244
24		2018-10-11	Disabled BIB command in makefile	2	3	3
25		2018-10-11	Added compile instructions	1	26	3
26		2018-10-11	README adjustments	1	3	3
27		2018-10-13	corrected name	1	1	1
28		2018-10-13	explain -localtime and -append root=/dev/vda rw console=ttyS0 debug	1	2	2
29		2018-10-13	add work log.	1	6	0
30		2018-10-13	Added the version control table	1	31	1
31		2018-10-13	Changed up how the lists were made	1	48	31
32		2018-10-15	Refactored rdrand to use X86 intrinsic and created a general rand function	1	88	69
33		2018-10-15	Updated concurrency writeup to reflect the refactored code.	1	7	10
34		2018-10-15	Updated work log	5	477	77
35		2018-10-15	Removed thirdparty	1	0	362
36		2018-10-22	Added concurrency2	2	219	0
37		2018-10-23	Setup writeup2 and update comments in concurrency	5	233	4
38		2018-10-27	Unignored Linux Yocto with our sched	7	2	1
39		2018-10-27	Unignoring yocto	1	0	2
40		2018-10-27	yocto...	1	1	0
41		2018-10-27	Added clook scheduler	4	582	2
42		2018-10-27	Setup Writeup2	11	9310	501
43		2018-10-27	Updated makefiles	3	8	9
44		2018-10-27	Making refs work	1	1	1
45		2018-10-27	Added Eyton's scheduler tester	2	16	0

Please continue at the next page

V	tag	date	commit message	MF	AL	DL
46		2018-10-27	Added David's makefile	2	27	1
47		2018-10-28	Fixed compile errors in CLOOK and modifed Kconfig for inclusion	4	84	6
48		2018-10-28	Added command for starting VM	1	5	0
49		2018-10-29	Made clook print if unexpected access order	2	8	0
50		2018-10-29	Clook Scheduler Writeup	11	165	8
51		2018-10-29	Fixes to README	1	7	7
52		2018-10-29	Version control log	1	79	0
53		2018-10-29	More work on Writeup2	18	101	91
54		2018-10-29	Updated makefiles and writeup for assign 1 and 2	13	69	9021
55		2018-10-29	Modified assignment 1 make	2	9	3
56		2018-10-29	Added the Concurrency Section 2	1	35	0
57		2018-10-29	Changed code styling	1	2	2
58		2018-10-29	Update writeup2.tex	1	21	0
59		2018-10-29	Minted Python Testing Script	1	6	7
60		2018-10-29	Update writeup2.tex	1	1	14
61		2018-10-30	Added raspberry linux to ignore file	10	1	1
62		2018-10-30	Started Writeup3	5	175	0
63		2018-11-07	Update README.md	1	4	0
64		2018-11-07	Created writeup for Downloading Raspberry Pi Linux Kernel, Preparing SD Card, and Testing Raspbian.	4	74	10
65		2018-11-07	Updated makefiles so that PDFs are copied to resulting folder	7	4	1
66		2018-11-08	Completed Part 1 for Writeup	3	52	21
67		2018-11-08	updated compress / extract commands in README	1	3	8
68		2018-11-10	Work on Writeup3	2	16	4
69		2018-11-10	Uploading modifid Linux files	7	712	0
70		2018-11-10	Finalized morse code trigger	2	38	34
71		2018-11-10	Writeup for compiling and running Morse code LED trigger	3	61	35
72		2018-11-10	Recompiled writeup3	1	0	0
73		2018-11-10	Fixed references	1	2	2
74		2018-11-10	Fixed formatting	2	11	17
75		2018-11-10	Updated PDF	1	0	0
76		2018-11-10	Added version control to PDF	2	105	0
77		2018-11-10	Added the finished writeup	1	47	7
78		2018-11-10	Recompiled PDF for writeup 3	1	0	0

Please continue at the next page

V	tag	date	commit message	MF	AL	DL
79		2018-11-10	Fixed typos	1	10	10
80		2018-11-10	Recompiled writeup3	1	0	0
81		2018-11-19	add 2nd concurrency part.	2	268	1
82		2018-11-20	Addded speeed	1	18	12
83		2018-11-22	Added writeup setup for concurrency 3 and assignment 4	6	677	49
84		2018-11-22	Fixed makefile and instructions for Writeup3	3	2	2
85		2018-11-22	fixed writeup4 compiling	2	54	0
86		2018-11-22	Work on char device	1	99	94
87		2018-11-22	Fixining includes	1	16	0
88		2018-11-22	Commenting out morse debug printk	1	5	5
89		2018-11-22	Fixed speed option	1	1	1
90		2018-11-22	Working on charachter device	12	180	91
91		2018-11-22	Finished message device	3	16	9
92		2018-11-22	Generated PDFs	2	0	0
93		2018-11-22	Fixed readme	1	1	1
94		2018-11-22	Unregistering char device for moment	1	0	4
95		2018-11-22	Added options to control repeat or one-off modes	3	132	85
96		2018-11-22	Fixed compile error	1	48	6
97		2018-11-22	Fixed another error	1	1	1
98		2018-11-22	Fixed one-off mode	1	13	9
99		2018-11-26	fix bugs and add features	1	33	14

References

[1] S. Monk, "Software installation," https://learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable/software-installation-mac, Nov 2018.

[2] ——, "Adafruit raspberry pi lesson 5. using a console cable," https://learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable/overvie Nov 2018.