Assignment 2
CSE 134: Embedded Operating Systems, Spring 2021

Due: Friday, April 30 at noon

Goals

The primary goal of this assignment is to implement a device driver that can take a string and "output" it as a sequence of light pulses, and can "read" a sequence of light pulses and return it in response to a read request. This will involve learning how to write a character device driver in FreeBSD, learning how to write and debug kernel code, and getting software to interface with hardware. You'll be working in your project team, so this is also a chance to learn how to collaborate, using `git` to exchange files and track changes and progress.

Basics

For this assignment, you'll work with your project group to implement a device driver that can send information via an LED laser and receive it via a photosensor. These devices will be connected to your Raspberry Pi using the GPIO pins.

The LED will transmit information using a sequence of short (dot) and long (dash) flashes of light. A dot lasts for $0.5X$–$1.5X$ ms, and a dash lasts for at least $3X$ ms, where X is the minimum length of time between flashes. The value of X may be set via `ioctl()` to the driver, and defaults to 20 ms. The driver sends and receives plain (ASCII) text, using a Huffman coding dictionary that we will provide on Canvas.

The driver must support both writing (flashing the LED using the code provided) and reading (reading the photosensor and decoding the information). Reading and writing may be done simultaneously, and may be done by the same or different processes.

Details

GPIO pins

You're writing a *character* (not block) driver that can send and receive a subset of ASCII characters via light. However, a Raspberry Pi doesn't directly control lights or sensors, but can interact with them via GPIO pins.

There are several sources of information you can leverage to learn how to interact with GPIO pins. There's a Python3 library that works on FreeBSD, `https://github.com/ethanlmiller/python-GPIOfbsd` (yes, written by Prof. Miller), that you can use to exercise the GPIO pins. You can also look at various GPIO software and drivers in FreeBSD for sample code and functionality.

The circuit for the LED and photodetector are very simple, and will be covered in lab section. You'll need a breadboard, cable to connect it to the GPIO pins, connection wires, and three discrete parts: an LED laser, a photodetector, and a pull-up resistor. These discrete parts can be purchased in bulk for about $10 on Amazon, and individual sets are available for a dollar from Prof. Miller or the TA. However, the breadboard and connection to GPIO pins should be purchased from Amazon; cost is about $10–15.

Device driver

Your group will be writing a device driver. Since this is code that runs *in the operating system*, there's no safety net. Bugs mean crashes, and crashes mean your OS kernel fails. This means that your code must *never* have undefined behavior. It's OK to refuse an operation you can't do for some reason. However, you must *always* check return codes and plan for what to do if you don't get what you expect.

Your device driver must implement `read()` and `write()` functionality, as well as handle `open()` and `close()` properly. It must also provide several `ioctl()` calls to provide a dictionary, retrieve the current dictionary, set the pins to use for the light and sensor, and set the minimum timing value X in milliseconds (see above). **Details on `ioctl` calls will be provided during lab section.**

Your driver must be able to buffer up to 8 KiB of incoming (ASCII) data if nobody is waiting for it. Note that this may correspond to a variable number of bits. It must also be able to handle blocking I/O, where a process requests (say) 20 bytes of data and is willing to wait for it.

Deliverables

Your driver must be in the form of a kernel module. The techniques for writing one will be covered in lab section—we *strongly* suggest that you attend section to learn about this.

Because your code will be in a separate module, you shouldn't need to check the entire kernel into `git.ucsc.edu`. Instead, you should just have a single directory corresponding to your module, and build the code on your local Pi as needed.

Your code must be written in an `asgn2` subdirectory, and must provide the `ledtext` driver module.

You must **commit** and push (to `git.ucsc.edu`) your design document (`DESIGN.pdf`) no later than Wednesday, April 21st.

Committing and submitting

You must do regular commits of your code using `git`; you'll lose points if you don't do regular commits. You need not push your commits to `git.ucsc.edu` every time you commit, but you should push on a regular basis. Please see the course Canvas page for details. Of course, `git` is the *best* way for your team to exchange code and other documents. You're encouraged to put the "source" for your design document into the `asgn2` directory, though we'll only grade a PDF design document named `DESIGN.pdf`.

Do not commit object files, assembler files, or executables to your `git` repository. Each file in the submit directory that could be generated automatically by the compiler will result in a 10 point deduction from your programming assignment grade.

You must follow the full submission instructions listed under Assignment 2 on Canvas. This includes the part about writing up your own contributions as well as those of the other members of your group. **Each member of the group must submit their own writeup of what they did and their impression of what others did.** In addition, each group member must submit the commit ID they wish to be graded.

Hints

- Start early!
- Meet with your team early to divide up responsibilities and discuss ideas. Meeting frequently early on will dramatically help you get a good design, allowing you to work more independently later on as you write code and debug it.
- Experiment with the sample programs and existing GPIO utilities to see how the systems work.
- Look at existing driver code to see how drivers are written. Make sure you cite any drivers from which you copy code. It's OK to do so—that's how you learn how to write drivers—as long as you cite your sources.
- Write your design document *before* starting to write your code. It's ok to experiment a bit to say how things work, but make sure you come up with a good design first, so you don't end up rewriting a lot of code.
- Get your design done by early next week (April 19–20). This will give your team more than a full week to write your code and debug it.
- Get your kernel development environment set up ASAP. Each team member should have their own development environment. You'll need to download the kernel source; you should clone it from a `git` mirror into `/usr/src`, since `/tmp` on a Pi is too small to use SVN.
- Work in small chunks and commit code often. Kernel development is *hard*. It's not unusual to write a 10–20 lines, commit them, and test them. If you broke something, you can easily see which lines of code might have been at fault.
- Get `write()` working first, then `read()`. `write()` will be easier. There's an existing `gpioled` driver that you can look at for ideas on how to get `write()` working. `read()` will be more complicated, and there isn't necessarily as good of a model to use for it.
- Leverage the collective knowledge and wisdom of your team if you get stuck. There's a reason you're doing this assignment in a group!
- If you have generic questions, ask them on Piazza.

Grading

Your grade for this assignment will be determined based on the rubric posted on Canvas. A well-done design document, good coding style, and good comments are all important to your overall grade, as is functionality of your driver. **The entire team gets the same base grade, with possible modifications if one or more members do *significantly* more or less work than everyone else.** Small differences in work are normal, and we *expect* that different team members will work different amounts on different parts of the assignment (design, coding, debugging, etc.). Our concern is over a team member who ghosts everyone else, or a team member who does the project themselves.

We can only grade what you commit and submit. We ***strongly*** recommend that, before making your submission, you clone your `git.ucsc.edu` repo to a new directory and try to build it. If you can't, figure out why not and fix it *before* submitting. **If you submit a commit ID that doesn't build with `make` with no arguments, your maximum grade on the assignment is 5 points**.