CSCI 8530: Computer Science
Advanced Operating Systems
Spring 2020
Program Assignment 3

Due on April 30, 2020

Introduction

The Xinu distribution named xinu-led.tar.gz on the *Odin* server contains several additional files. The files **system/gpio-i2c.c**, **include/gpio-i2c.h**, and **include/galileo_gen_defs.h** (with the additional of a few **typedefs** in **include/kernel.h**) provide the tools necessary to control the state of a light-emitting diode (LED) on the Galileo Gen 2 board; the particular LED is located immediately adjacent to the USB host connector. There is also a new shell command in the file **shell/xsh_led.c**, and the appropriate changes have been made to **shell/shell.c** and **include/shprototypes.h**. This new command demonstrates the ability to "blink the LED." Blinking an LED is often viewed as the "Hello, world" program for embedded systems.

In this assignment, you shall add a new device named **LED** to the Xinu system, including the appropriate device driver functions, and a demonstration program **shell/xsh_led_demo.c** that has the same functionality as **shell/xsh_led.c**, but accomplishes the work using the new LED device driver.

The Device Driver

Your device driver will deal with only a single LED (or digital General Purpose Input/Output, or GPIO, as they're usually called in the embedded world). Normally we would also want to allow digital input on a GPIO connection, but we'll not attempt that in this assignment. However we will want to be able to determine the state of the LED (that is, on or off).

As you can see by examining almost any of the existing device drivers (in the various subdirectories of **device**), there is usually a separate file for each of the following system calls for a device:

init, open, close, getc, putc, read, write, and control

There may also need to be an interrupt handler, and functions to deal with different types of interrupts for a device. For the LED we'll not be concerned with the interrupt handler.

1. The **init** function for the LED device will be invoked as part of the system startup procedures. If you examine the **xsh_led** program's source code, you'll see the hardware initialization actions that need to be completed in the **init** function.
2. The **open** function for the LED device will be very simple. It should verify that the device (the LED) is not already open; if so, it should return **SYSERR**. If the device is not open, it should save an indication that it is now open, set the device state to off (that is, not illuminated), and return the device number associated with the device.
3. The **close** function should be equally simple. If the device isn't open, return **SYSERR**. Otherwise make certain the LED is not illuminated, and mark the device as not open.
4. The **getc** function will return one of three results: **SYSERR** if the device is not open, 'Y' if the LED is "on" (that is, illuminated), and 'N' if the LED is "off" (not illuminated).
5. The **putc** function has two arguments: the device identification[1] and a single char, which should be 'Y' or 'N'. If the device is not open, or if the char argument is not 'Y' or 'N', putc will return **SYSERR**. Otherwise the LED state should be set to "on" or "off," as appropriate.
6. Both of the **read** and **write** functions for the LED device should return **SYSERR** if they attempt to read or write more than one character, or if write attempts to write anything other than 'Y' or 'N'.
7. The **control** function for our LED device has no defined work to do, so it should always return **SYSERR**.

All the functions for the LED device should begin with **led** (for example, **ledopen**, **ledclose**, ...) and they should be placed in a new **device** subdirectory named **LED**. Also note that you will need to modify the **config/Configuration** file to specify the "type declaration" for the new device (in the first part of the file), and an actual declaration for the LED device (in the second part of the file – after the first **%%** marker).

The xsh_led_demo.c Program

To demonstrate the proper operation of your device driver, write a new shell program that has exactly the same functionality as the **xsh_led.c** program (including the option of varying the blinking period).

Before starting the blinking, though, your program should include code that verifies each of the various operations that could fail (returning **SYSERR**) does return the proper result. For

[1] Of course the device identification must be passed as the first argument to all of the functions except **open**; **ledinit** is passed a pointer to the device's entry in the device switch table.

example, you'll want to try opening the LED device when it is already open to verify **SYSERR** is return in this case. Also check to see that **close** applied to the LED device when it is not already open returns **SYSERR**. Obviously try something other than 'Y' or 'N' as the character to be written using the **putc** function, and verify that **getc** returns the proper result. Note that the list given here is **not** exhaustive. There should only be output from the checking in the event of an error.

Submission

To submit your work, create a directory on Loki named **/home/<u>myusername</u>/ csci8530-prog3-s20**, replacing **<u>myusername</u>** with your username. For example, if your username was **phuang**, then the directory you create should be named **/home/phuang/csci8530-prog3-s20**. Put all of the source code (that is, the files) in that directory, **and nothing else**. During testing of your work, it will be assumed that **xsh_led_demo.c** will be put in the **shell** directory (you do not need to include the modified **shell.c** or **shprototypes.h** files), **Configuration** will be put in the **config** directory, and all other C source code will be put in the **device/LED** directory. If you include other files, or you require some files to be placed in different directories, then include a **README** file that explicitly states your requirements.
Include comments in your code that explain what's happening. That is, an individual reading your code should be able to understand its purpose and the procedure used to achieve that purpose. Turn off, disable, or remove any debugging code you may have used before submitting your work.
Make certain you include – in each file – a comment near the top of the file that includes the name or names of each author of the work and **<u>also snapshot the results from the console and record a video of the LED blinking</u>** after porting the codes to the board. If there are multiple authors, submit the joint work, only once, through **<u>a single submission directory</u>**. For example, if there are two members in your group, only ONE of these individuals should have a submission directory.