ECE531: Advanced Operating Systems – Homework 3

Device Driver for the Raspberry Pi Serial Port

Due: Friday, 22 September 2023, 5:00pm

This homework is about communicating to your bare-metal Pi system via a serial port.

1. **Download the homework code template**

   - Download the code from:
     `https://web.eece.maine.edu/~vweaver/classes/ece531/ece531_hw3_code.tar.gz`
   - Uncompress the code. On Linux or Mac you can just
     `tar -xzvf ece531_hw3_code.tar.gz`

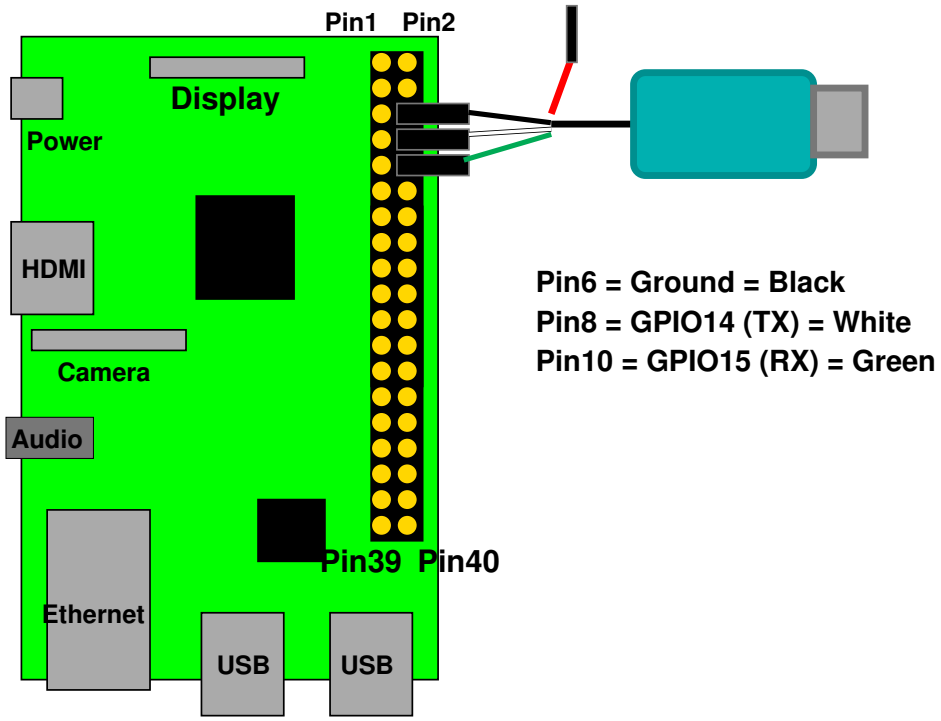

Figure 1: Raspberry Pi Serial Connection (Pi3 shown, Pi4 is similar)

2. **Get serial input/output working**

   - Connect the serial to USB converter to your Pi as shown in Figure 1.
   - It is not necessary to connect the red wire (5V) to the Pi. It provides 5V that you could use to power the pi, but it bypasses the power filtering and can cause bad things to happen if you are already providing power to the pi via the normal USB-micro or USB-C ports.
   - Additional details for using the adapter can be found here:
     `http://www.adafruit.com/products/954`
     specifically:
     `https://learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable`
     If you are connecting to a Windows machine you may need to download the PL2303 driver as described in that link You might also need to follow the directions to get things working on MacOS. Linux should come with the PL2303 driver by default.

- Setup a terminal program on your development machine.
  - We will program the Pi to use settings 115200 8N1 and software flow control.
  - For Windows I recommend "putty". See the adafruit link for download and usage info. There is also some info at the end of this document.
  - For Linux, minicom works. You might need to install it (`apt-get install minicom` on Debian or Ubuntu).
    Plug in the device then look at the end of the `dmesg` command to see what device it is. (Usually `/dev/ttyUSB0`).
    Run minicom with something like:
    `minicom --color=on -D /dev/ttyUSB0`
    You may need to be root. Menus are accessed by **control-A** then **Z** for help. You might have to disable hardware flow control (**control-A O**. It might also help to enable line wrapping (**control-A W**).
  - MacOS. The adafruit link above has info on how to get this working using the `screen` program, there's some additional information at the very end of this document.

3. **Familiarize yourself with the provided code**

   I provide the following files:

   - `bcm2835_periph.h` – useful #defines based on the Broadcom 2835 Peripherals manual
   - `boot.s` – assembly boot code that sets up the stack and clears the BSS
   - `console_io.c`, `console_io.h` – common entry point for console printing code
   - `delay.h` – inline assembly for creating delay loops
   - `kernel.ld` – linker script
   - `kernel_main.c` – the main entry point to the code
   - `mmio.h` – inline assembly for doing I/O accesses
   - `printk.c`, `printk.h` – code for the printk routine
   - `serial.c`, `serial.h` – the UART code
   - `led.c`, `led.h` – code to enable GPIO18 LED

4. **Get your serial port working (2pt)**

   - First edit `kernel_main.c` at the top and uncomment the proper `IO_BASE` value for the type of Pi you have.
   - Edit `uart_init()` in the `serial.c` file. Much of this is already implemented for you.
   - You will need to add some MMIO calls. Unlike the examples shown in class, use routines called `bcm2835_write()` and `bcm2835_read()`. These routines take into account the IO_BASE value.
   - Modify the code so it properly sets the `UART0_IBRD` and `UART_FBRD` values for 115200 operation. We went over the calculations on how to do this in class.
     Use 48MHz as the UART clock.
   - At the end of the function be sure to set the `UART0_CR` register to enable the UART overall, transmit, and receive. (See class notes for more details)

- Compile the code, using `make`
- Copy the generated `kernel7.img` or `kernel7l.img` to your SD card and boot it as per HW#2 (remember `kernel7.img` is for Pi2/Pi3, `kernel7l.img` is for Pi4).
- If all goes well you should be able to type things in your terminal and have them echoed back to you with the letters changed to uppercase. (Note, pressing Enter might not work as expected because of the linefeed/carriage return issue discussed in class).
- Note at boot that it will prompt you to press a key before continuing. This is a bit of a hack; without it if you are using screen or putty and also powering your board from the serial port then sometimes the boot messages will scroll by before the terminal program is ready to display them.
- If you still have an LED connected to GPIO18, then it should light up if your uart init code returned (the code to do this is in `led.c` and called after the uart init in `kernel_main.c`). You can use this routine to help debug things as a last resort.
- Getting the serial port working is critical, so if you get stuck here please ask for help right away.

5. **Print a message at bootup (2pt)**

   - Modify the `kernel_main.c` file to print a boot message of your choice.
   - You can do this simply by calling the provided `printk()` function with your string to print. `printk()` is just the kernel version of `printf()`
   - Remember you might need to have a CR/LF line ending (`"\r\n"`) to get a newline.

6. **Get printk printing hex values (2pt)**

   - Right after your boot message we print the hardware type, which can be found in the `r1` variable.
   - We want to use `printk()` to do this in hexadecimal, but `printk()` as provided does not properly support the %x modifier.
   - Fix the code in `printk.c` so the %x modifier works.
   - Add a printk statement in `kernel_main.c` to print the value in hex.

7. **Something Cool (1pt)** Be sure to describe what you did in the README file.

   Suggestions of things you can do:

   - Clear the screen (using escape characters) at boot.
   - Print a fancy boot message that is in color and has ASCII art.
   - Interpret the keys being pressed and do something interesting (turn on/off the LED, change the text color, etc.)
   - Add other formats to printk (%c, %s, %X, etc).

8. **Answer the following questions (3pt)**

   Put your answers in the README file.

   (a) Why do we use the serial port for communication with the Pi in this homework rather than USB, HDMI video, or ethernet?

   (b) What are parity bits good for in a serial connection?

   (c) What is inline assembly language?

   (d) A common way to write a simple command interpreter is to use the standard C `strtok()` function. Why don't we use that in this homework?

   (e) Which terminal program and operating system did you use for this homework? Is it working well or are there annoying issues with getting it working?

9. **Submit your work**

   - Run `make submit` in your code directory and it should make a file called `hw3_submit.tar.gz`. E-mail that file to me by the deadline.
     Be sure your name (and your partner's name if applicable) is in the file!

Some extra notes on getting serial ports working

- MacOS
  - Generally the linked tutorial to the Aafruit page should tell you what you need to know, be sure to keep clicking next to get past the Linux directions
  - There are two drivers you might need to install. The old one (prolific) can be downloaded here, as the link from the adafruit page appears broken. `https://www.prolific.com.tw/US/ShowProduct.aspx?p_id=229&pcid=41`
  - If you have one of the newer cables you might need the Silicon Labs driver instead
  - To see if you have the proper driver installed, be sure the USB-serial device is connected, open a terminal and do a `ls /dev/cu*`. If things are working you'll see a file called something like `cu.usbserial` or similar (it might have random hex digits after it).
  - Once you see that file, you can use the `screen` tool to connect to the serial port. Something like:
    `screen /dev/cu.usbserial 115200`
    where cu.usbserial is the serial port you saw in the previous step.
- Windows
  - For windows again the Adafruit directions should be more or less enough to get you going.
  - I did hand out both models of serial cable, so you might have to install both the Prolific and the Silicon Labs drivers.
  - The prolific link might not work in anymore, you might want to try this one instead: `https://www.prolific.com.tw/US/ShowProduct.aspx?p_id=229&pcid=41`
  - Ideally windows would auto-install the driver. If not, you should get an EXE file to click on when you download the drivers. If there is no EXE, you might have to right-click on an .inf file and pick "install"
  - Check the device manager and hopefully a USB serial connection has appeared. It should appear with a name something like COM7
  - Start putty and use the COM name from the previous step. Be sure you are connecting at 115200 bits per second