Talking to a Serial Port

ECE598: Advanced Operating Systems – Homework 2

Spring 2015

Due: Wednesday, 11 February 2015, 5pm

This homework is about communicating to your standalone program via a serial port.

1. **Download the homework code template**

   - Download the code from:
     `http://web.eece.maine.edu/~vweaver/classes/ece598_2015s/ece598_hw2_code.tar.gz`
   - Uncompress the code. On Linux or Mac you can just
     `tar -xzvf ece598_hw2_code.tar.gz`

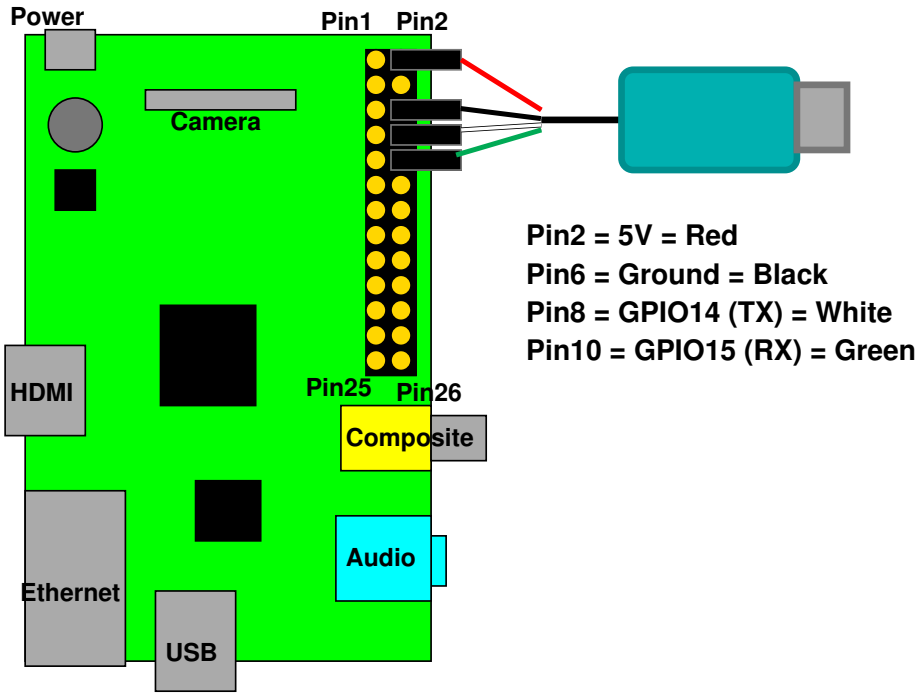

Figure 1: Raspberry Pi Serial Connection

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

   - Connect the serial to USB converter to your Pi as shown in Figure 1. For the model B+ the connectors are similar (the GPIO header is longer but the top half of the pins are the same).
   - When hooked up this way, you do *not* need to use a separate USB/power cable to power your Pi. The Pi can get power from the USB/serial connector.
   - 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. The same goes for OSX. Linux should come with the PL2303 driver by default.

- Setup a terminal program on your development machine.
  - For Windows I recommend "putty". See the adafruit link for download and usage info.
  - 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 control-Z for help. You might have to enable sw flow control. It might also help to enable line wrapping (control-A control-W).
  - OSX. There are terminal programs out there, I haven't had a chance to find a good one. The adafruit link recommends screen, which will work but you'll miss your initial boot messages unless you put a long delay at the start of your OS so you have time to insert then run the command.

3. **Get your serial port working (1pt)**

   - Edit the `serial.c` file. Look for the "Your Code Here" comment. You will need to pick the values for the UART0_IBRD (integer value of baud rate divisor) and UART_FBRD (fractional value of baud rate divisor). There are comments there saying how to calculate. Write the proper value to each register. You may have to round the values.
   - Compile the code, using `make`
   - Copy to a memory key and boot it as per HW#1.
   - If all goes well you should be able to type things in your terminal and have them echoed back to you.
   - Getting this working is critical, so if you get stuck here be sure to ask for help right away.

4. **Print a message at bootup (1pt)**

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

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

   - Next we want to print the hardware type, which is stored in variable r1. You can use printk with `"%x"` to print this value.
   - You'll note that the value is printed in decimal. We want hex. So open `printk.c` and find the code that is responsible for printing hex values and modify it to print a hex value instead.

6. **Parse and Print the ATAGS (2pt)**

- We are interested in knowing the total RAM available for our use. This can be found via the ATAGs copied to us by the bootloader. The format for these is an array of data items where 32-bit int (size), 32-bit int (type) then chunks of 32-bit ints until the size is met. We want to parse this list looking for the `ATAG_MEM` value, and then print that value to the screen using printk()
- Modify the dump_atags() routine in `atags.c` to print the memory size. It is found in the first value after the ATAG_MEM type.

7. **Something Cool (1pt)** Suggestions of things you can do:

   - Clear the screen (using escape characters) at boot.
   - Print a fancy boot message that is in color.
   - Add code that manipulates the ACC LED at useful times.
   - Add code that prints the ATAG cmd line.

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

   Put your answers in some sort of document (.txt, .pdf, .doc).

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

   (b) What are parity bits good for?

   (c) By default most systems these days use settings of 115200 bits/second, 8-bits, no-parity, 1 stop bit. Why is parity disabled?

   (d) What is inline assembly language?

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

   (f) 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 `hw2_submit.tar.gz`. E-mail that file to me as well as the document with the answers to the questions.