CMSC 328: Embedded Operating Systems

Serial Communication with the Pi

Our goal in this lab is to extend bbOS to communicate with another computer (e.g. Ubuntu Desktop, a Mac, PC or another Pi) using serial communication. By the end of the lab, we will have a small client-server system that allows programs running a host machine to control the inputs and outputs of the Pi.

Completing the UART in bbOS

The bbOS from last lab is able output text using `putc()` and `puts()`, but it can't receive input.

1. Add two functions to **uart.c**
    a. `int uart_getc()` returns a byte of data; -1 if no data is available
    b. `int uart_getcblock()` return a byte of data; blocks until there is data

   The `UART0_FR` and `UART_DR` registers are used for this.
   (see Ch. 13 of BCM2385 Peripherals Manual[1])

Inputs in bbOS

Modify bbOS to read a button attached to GPIO 23 and blink GPIO16 if pressed.

1. Connect a button to the breadboard such that it spans both halves of the breadboard. Each leg will be on its own line of the breadboard.
2. Connect one leg of the button to pin 23 and the other leg (on the same side) to GND.
3. When the button is pressed GPIO23 will be connected to GND, so we'll use a pull-up resistor to set GPIO23 high when floating (i.e. when the button is not pressed).
    a. Read `[GPLEV0]` to inspect the state of the pin.
    b. Write to `[GPPUD]` to enable this Pull Up Register (Ch. 6 of BCM2385 PM)

Client/Server

1. Write a small server program that runs on the PI. It should support three commands 'B' 'N, and 'F'. When the server program receives:

"N"	the server should turn GPIO 25 on
"F"	the server should turn GPIO 25 off
"B"	the servers should respond with "1" if GPIO23 is high and "0" if low

4. Write a small client program on the PC (or your friend's Pi) using Python and pyserial[2] that sends commands to the Pi over the serial port. It should turn the LED on or off if the button is pressed.

[1] http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf
[2] http://pyserial.sourceforge.net/

Linux Kernel Modules

Switching gears a bit, next, we will use a linux kernel module to talk directly to the Pi's hardware. We'll have to recompile the Linux kernel and create two Linux kernel modules. Unfortunately, building kernel modules with the 3.6.11+ kernel on the RPi fails, so we will have to use a cross-compiler to build the kernel and modules somewhere else and then copy the **kernel.img** and **lib/modules** and **lib/firmware** to the Pi afterward.

Building the Linux Kernel

1. Find your Pi's IP address (referred to as `10.10.RPI.IP` later) using `/sbin/ifconfig`
2. Back up your kernel.img and kernel modules on the PI
   ```
   pi$ cd /
   pi$ tar -cvzf ~/backup.tgz boot/kernel.img lib/modules lib/firmware
   scp pi@10.10.RPI.IP:backup.tgz .
   ```
3. On Ubuntu, build a custom linux kernel[3]:
   a. Install the cross compiler and supporting tools:
      ```
      git clone git://github.com/raspberrypi/tools.git
      ```
   b. Get the linux kernel source:
      ```
      wget https://github.com/raspberrypi/linux/archive/rpi-3.6.y.tar.gz
      ```
   c. Unpack the linux kernel source:
      ```
      tar -xvzf rpi-3.6.y.tar.gz
      ```
   d. Move into the linux directory
      ```
      cd linux-rpi-3.6.y
      ```
   e. Grab the `.config` (kernel configuration on pi) and put it in the linux directory
      ```
          scp pi@10.10.RPI.IP:/proc/config.gz .config.gz
          gunzip .config.gz
      ```
   f. Setup the CCPREFIX variable to point to the cross-compiler
      ```
      export CCPREFIX=~/tools/arm-bcm2708/arm-bcm2708-linux-gnueabi/bin/arm-bcm2708-linux-gnueabi-
      ```
   g. Build the kernel (this will take a while, move on to the next part of the lab)
      ```
      make ARCH=arm CROSS_COMPILE=${CCPREFIX}
      ```
   h. Make a directory to hold the kernel modules (e.g. ~/modules)
      ```
      mkdir ~/modules
      make ARCH=arm CROSS_COMPILE=${CCPREFIX} INSTALL_MOD_PATH=~/modules modules_install
      cd ~/modules
      tar -cvzf libfirm.tgz .
      ```
   i. Copy the kernel.img and ~/modules to the pi
      ```
      scp kernel.img pi@10.10.RPI.IP:
      scp ~/modules/libfirm.tgz pi@10.10.RPI.IP:
      ```

[3] http://elinux.org/RPi_Kernel_Compilation

4. On the Pi install the kernel modules and the kernel image:
```
pi$ sudo cp kernel.img /boot/kernel.img
pi$ cd /
pi$ sudo tar -xvzf ~/libfirm.tgz
```
5. Reboot and run the new kernel!
```
pi$ sudo /sbin/reboot
```

Building a Linux Kernel Module

1. Download and build the provided kernel module.
    a. **rpiheart** uses a timer to blink the LED as a sort of heartbeat.
2. Load the module and view the debug messages from `printk` using `dmesg`.
```
pi$ sudo insmod rpiheart.ko
pi$ dmesg
pi$ sudo rmmod rpiheart
```
3. Answer the following questions about how **rpiheart** works:
    a. What are the parameters to `gpio_direction_output` used for?
    b. Why is `hrtimer_forward` called in the timer callback?
    c. Useful resources on kernel modules, gpio and timers:
    http://www.tldp.org/LDP/lkmpg/2.6/html/
    https://www.kernel.org/doc/Documentation/gpio.txt
    http://www.ibm.com/developerworks/linux/library/l-timers-list/

Deliverables

lastname_lab7/

uart.c	UART implementation
main.c	RPI server
client.py	A python program (using pyserial) that controls the Pi via serial.
lab7.txt	Answers to the questions about the linux kernel modules.

Notes

The UART (i.e., serial port) is useful for debugging and communication.

http://learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable/overview
http://elinux.org/RPi_Serial_Connection

On Mac you can use the programs screen and zterm to access a serial port.
On Linux you can use the programs screen or minicom to access a serial port.
On Windows you can use putty to access a serial port.

Extra

1. Write the serial client in C rather than in Python.