CMSC 328: Embedded Operating Systems

bbOS: Bard Blinker OS

The Raspberry Pi normally runs the Linux kernel, but that is not the only way in which it can be used. In this lab, we'll write our own OS that runs on the Pi. Our OS won't do much at this point, we will start by blinking the LED (Lab 1) without the aid of the Linux Operating System. In order to accomplish this, we will need to use some low-level assembly programs and linking scripts.

Tutorials

1. Make a copy of the **kernel.img** file on your SD card. We will be overwriting this file during the lab – **keep a backup copy.**
2. Although we can use the ARM compiler (`gcc`), assembler (`as`) and linker (`ld`) on the Pi for this lab, it is will be convenient for you to install a cross-compiler using the GNU ARM toolchain. Cross-compilers let us compile ARM code from an Intel-based PC.
   http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/downloads.html
3. Complete the first three tutorials *OK01–OK02*.
   http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/index.html
4. Modify the *OK02* tutorial to toggle GPIO25 rather than GPIO 16. We used GPIO25 in our first lab to blink an LED. Chapter 6 of the BCM Peripherals Manual will have the information for how to control the directionality of the GPIO pins, as well as how to set and clear them. This manual also details the other capabilities of the BCM2835 processor (e.g. UARTs, interrupts, timers, etc.); it is worth downloading.
   http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf
5. Complete tutorial *OK03*.
   http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/ok03.html
6. Download, compile and test bbOS; bbOS.tgz is available on moodle. After building it, you can copy the kernel.img file to your SD card. The green ACT light should blink.
7. Modify bbOS to blink the LED connected to GPIO 25 rather than GPIO 16. For example, to mimic the behavior of OK01, the following C snippet can be used:

   ```
   volatile unsigned int* direction = (unsigned int*)(0x20200004);
   volatile unsigned int* clr16 = (unsigned int*)(0x20200028);
   *direction = *direction | (1 << 18);
   *clr16 = (1 << 16);
   ```

   The left shift (<<) and or (`|`) operators are used to select the 16th bit in the 32-bit word.
8. Use `objdump` to disassemble the compiled C program. Compare it with the assembly you wrote in #4.
9. Complete either the *Screen01* or *Input01* tutorials.
   http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/screen01.html
   http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/input01.html

Deliverables

lastname_lab6/	
main.s	ASM-based GPIO 25 blinker
main.c	C-based GPIO 25 blinker
lab8.pdf	Write an explanation of boot.s.
	Why are the functions in MMIO useful? (note: they use inline assembly[1].)

Notes

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

- Call the assembly function you created in step #4 from within bbOS.
- Access the Screen or Input (step #9) from your C-based kernel.

[1] http://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html