Project #2 – Design and Implementation of Device Drivers

CSE321: Embedded and Realtime Operating Systems   **Fall 2011**

1. Objectives and overview of the project:

In Project #1 we prepared a simple hardware system and embedded in it a minimal operating system and added shell commands. In this project we will ass device driver. Objectives of the project are to:
(i) study the software representing various components of the Embedded XINU operating system
(ii) learn and understand the requirements for writing a device driver, and
(iii) add a simple device driver to enhance the capabilities of the Embedded XINU.

2. Project Environment:

2.1 Operating system XINU: XINU ("Xinu Is Not Unix") is a small, academic operating system developed at Purdue University by Dr. Douglas E. Comer in the early 1980s for the LSI-11 platform; it has now been ported to a variety of platforms. Embedded XINU is an update of this project which attempts to modernize the code base (to ANSI-compliant C) and port the system to a modern architecture (specifically the MIPS architecture).

2.2 Hardware WRT54GL: We will use wireless router WRT54GL as a host for embedded the software we will develop for enhancing the features of an embedded operating system. The WRT54G is notable for being the first consumer-level network device that had its firmware source code released to satisfy the obligations of the GNU GPL. This allows programmers to modify the firmware to change or add functionality to the device. WRT54GL features a Broadcom MIPS processor BCM5352 (200-250Mhz), a four port switch, 802.11b and 802.11g wireless LAN support, a Web interface for configuration of the router, 16Mbytes of RAM and 4Mbytes of flash memory. Any modification to the router function itself has to be done by updating the firmware on the flash memory. However we will use the RAM to load the embedded XINU.

2.3 Test Environment: The lab environment will be supported by other hardware and software components to provide cross-compiling and networking support. In order to compile Embedded MIPS kernels on a workstation that is not itself a MIPS processor, it is necessary to build and install an appropriate cross compiler. Any time you modify the embedded XINU, the recompiled software will have to be reloaded. A simple network configuration allocating IP address to the host with the cross-compiler, the wireless router that will host the embedded XINU and a common router that forms the network are shown in Figure 1.

IP: 192.168.0.101   **IP: 192.168.0.130**
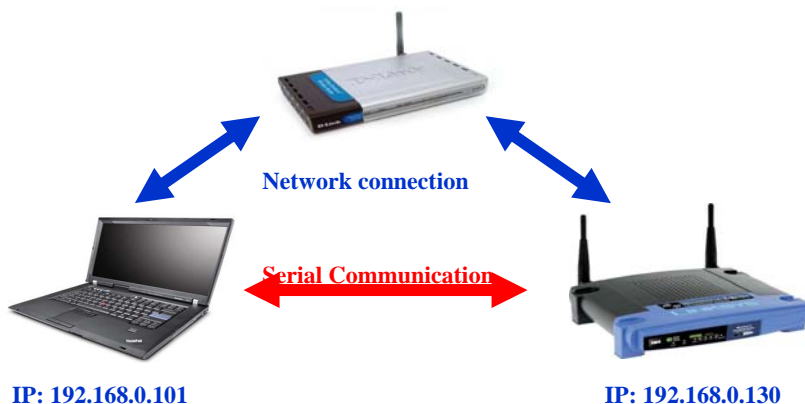
Figure 1: Network to enable uploading Embedded XINU into WRT54GL (by D. You)

The Common Firmware Environment (CFE) is the firmware developed by Broadcom for the BCM947xx SoC platform (among others). It is the first code that runs when the router boots and performs functions:

- Initializes the system
- Sets up a basic environment in which code can run
- Optionally provides a command line interface non-standard usage
- Loads and executes a kernel image

So, in normal operation, a user will not see CFE working at all; it will load the LinkSys kernel and exits. For us, however, CFE is crucial, because it provides us with the ability to load an image over the network using TFTP.

3. Device drivers

A device driver is a software module that abstracts a physical (hardware) or virtual devices and provides a uniform (and sometimes standard) interface for an operating system to access the peripherals of a system. A **framebuffer** is a video output device that drives a video display from a memory buffer containing a complete frame of data. The information in the buffer typically consists of color values for every pixel (point that can be displayed) on the screen. Color values are commonly stored in 1-bit monochrome, 4-bit palettized, 8-bit palettized, 16-bit highcolor and 24-bit truecolor formats. [http://en.wikipedia.org/wiki/Framebuffer]. Below we simulate the monochrome framebuffer and single pixel represented by an asterisk '*' character. Assume a suitable size for the buffer (say, 256characters X 256 characters). Consider for example a display window (16 X16), try to draw your initials using asterisk or any other character. In your project this frame buffer will be further enhanced.

			*												
		*		*											
	*				*										
*						*									
*						*									
*						*									
	*	*	*	*	*										
			*												
			*												
			*												

3. What to do?

You will implement the project in two phases:
Phase 1: Design and implement a simple user level framebuffer in order understand the function of a framebuffer. (20%)
Phase 2: Design and implement a framebuffer device driver that will enhance the xinu operating system with a device driver similar to tty driver. (80%)

3.1 Study Embedded XINU files

Embedded XINU documentation for the version of XINU we are using is available at
http://www.cse.buffalo.edu/~bina/cse321/fall2008/xinudocs/index.html . This documentation is internally well
linked to allow navigation among related code files. Examine the directory structure and understand the purpose
of various directories. Specifically, study these files in the directories:

- **include (header files)**: **shell.h, device.h , interrupt.h, tty.h, uart.h**
- **shell (shell command files): shell.c, xsh_led.c, xsh_help.c**
- **system (system operations): devtable.c, getc.c, putc.c, read.c, write.c, initialize.c, kprintf.c, open.c, close.c, startup.S**
- **all the files in tty and uart directories.**

3.2 Write a framebuffer driver (Phase 2)

The **framebuffer** is a graphic hardware-independent abstraction layer to show graphics on a console without
relying on system-specific libraries or the heavy overhead of a Window display system. The framebuffer driver
will be implemented as a virtual device on top of uart device driver. The tty driver is indeed a virtual driver that
is implemented on top of uart driver. Study tty driver to get an idea of the approach. For the framebuffer device
driver suggested steps are as follows:

1. Update **device.h and devtable.c** to include a framebuffer driver.
2. Create a **framebuf.h** that specifies the operations of the framebuffer: **init, control, open, close, write (or putChar), other functions needed.**
3. Create a directory **framebuf** and implement all the functions.
4. Update **Makefile** in the **compile** directory to include instructions to compile and link framebuf driver to the xinu boot.
5. Update other files such as the **initialize.c**, if needed.
6. Load the xinu boot and test the operation of the framebuffer driver.

3.3 More Details and Specification (prepared by Panya)

CSE 321 Project 2 (Fall 2011)

Objectives

Once you have completed this project, you will know how to:

1. Handle command-line arguments.
2. Parse command-line options
3. Use memory-related function calls, including `memset` and `free`
4. Use bit-wise operators
5. Set the background and foreground colors of your terminal

Requirements

The user should be able to input the command `fb` with the following options:

```
fb [-w width] [-h height] [-b color] [-f color] [-e effect] string
```

Options

[**-w** *width*]   Specify the width of the frame buffer. That is, the number of characters stored in each row of the frame buffer. The minimum value for *width* is 1 and the maximum is 8. The default value is 5.

[**-h** *height*]   Specify the height of the frame buffer. That is, the number of characters stored in each column of the frame buffer. The minimum value for *height* is 1 and the maximum is 8. The default value is 5.

[**-b** *color*]   Specify the color of the background of the frame buffer: the color of the terminal background. The default value is *white*.

[**-f** *color*]   Specify the color of the foreground of the frame buffer: the color of the characters. *color* can take any of the following values: *black, red, green, yellow, blue, magenta, cyan, white*. The default value is *black*.

[**-e** *effect*]   Specify the effect of the characters. *effect* can take any of the following values: *normal, underscore*. The default value is *normal*.

string   String to be stored and displayed in the frame buffer. This argument is required.

For instance,

```
fb –h 9 –w 6 –b yellow –f black –e underscore hello
```

Create a frame buffer with 9 rows and 6 columns. The color of the character is black and the color of the background is yellow. The format of the character is <u>underscore</u>.

For more information on color codes, visit http://ascii-table.com/ansi-escape-sequences.php.