Lab3 – Design and Implementation of Device Drivers

CSE524: Realtime Operating Systems **2013**

1. Objectives and overview of the project:

In this lab we will add a device driver. Objectives of the lab 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.

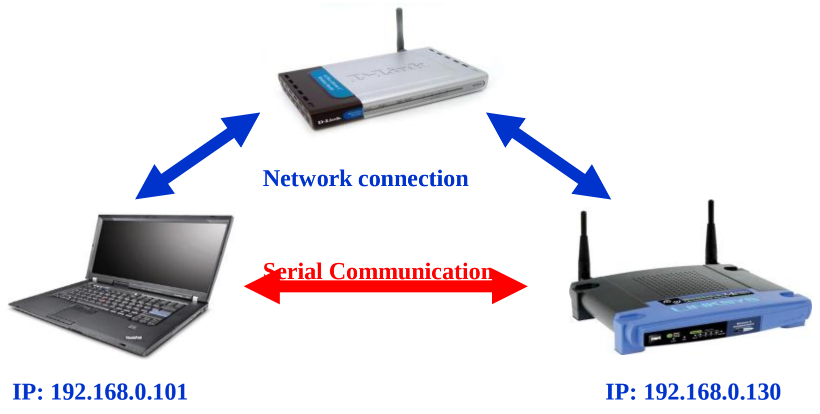

Figure 1: Network to enable uploading Embedded XINU into WRT54GL (by D. You)

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]. We will 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.

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

3. What to do?

You will implement the project in two phases:
Phase 1: Design and implement a simple user level framebuffer as **shell command** 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 as a shell command (Phase 1)

See http://xinu.mscs.mu.edu/Shell on how to add to the shell commands.

3.3 Write a real Framebuffer driver

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.

What to submit?

Zip the phase1 files as yourNameFBPhase1.zip

Zip the phase 2 files as yourNameFBPhase2.zip

submit_cse524a the zip files separated by space