NEXOS[1] Project #2 – Interacting with Embedded XINU Operating System

CSE321: Embedded and Realtime Operating Systems Fall 2014
Due date: 11/7/2014 (you need to demo it by then by attending recitations)

1. Objectives and overview of the project:

We have prepared a simple hardware system and embeded in it a minimal operating system. The minimal operating system will be enhanced in this project by other features. We will use WRT54GL the wireless router from Linksys [1] and Embedded XINU operating system [2] for this purpose. Students will

1. Study the architecture of WRT54GL wireless router.
2. Test the embedded operating system using simple C programs.
3. Modify the EXINU code to add new shell commands.
4. Study the software representing various components of the Embedded XINU operating system
5. Study the serial port features (UART0 and UART1) offered by XINU/WRT54GL platform
6. Establish a communication channel between devices connected to these ports via a cyclic executive loop and try out some priority inversion problem/solution.
7. Build a simple chat application with user login feature and realtime constraints.

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 the software we will develop for enhancing the features of an embedded operating system. WRT54GL [8] has a very interesting history the details of which can be found at [3]. 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 Ethernet switch, 802.11b and 802.11g wireless LAN support, a Web interface for configuration of the router, 16Mbytes of SDRAM 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 load the embedded XINU image that we create in the SDRAM. See [4] more details on the memory layout of the SDRAM.

[1] NEXOS: Next Generation Embedded and Realtime Systems project partially funded by NSF DUE-CCLI-0737243

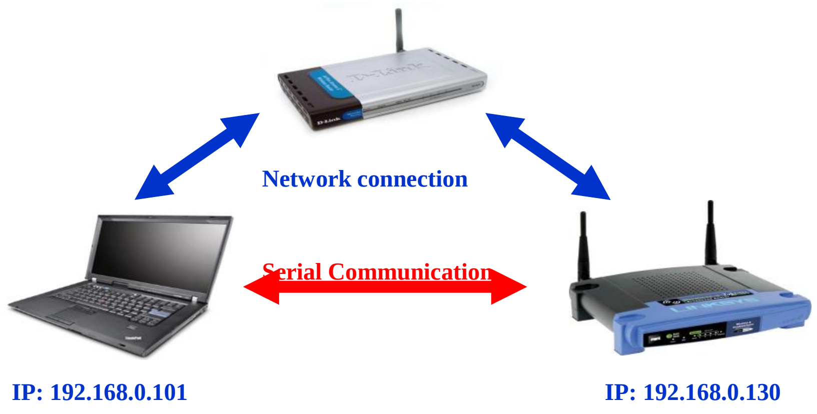

Figure 1 Basic NEXOS Network to Upload Embedded XINU to WRT54GL

2.3 Basic NEXOS 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 into the WRT54GL system you are working on. A simple and basic 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.

The Common Firmware Environment (CFE) [5] 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 send it on its merry way without hesitation. For us, however, CFE is crucial, because it provides us with the ability to load an image over the network using TFTP [6].

2.4 Expanded NEXOS Test Environment

The WRT54G can be used alone at any workstation with a serial port and a network connection. This configuration may be suitable for very small laboratory installations concerned only with hardware systems assignments, but to take maximum advantage of the power of our platform it is far more useful to collect many WRT54G's as a pool of *backends* doled out by a central server. Particularly in the context of more advanced courses in networking, it is best that this pool of backends be managed on a private network, isolated from the production network by a gateway.

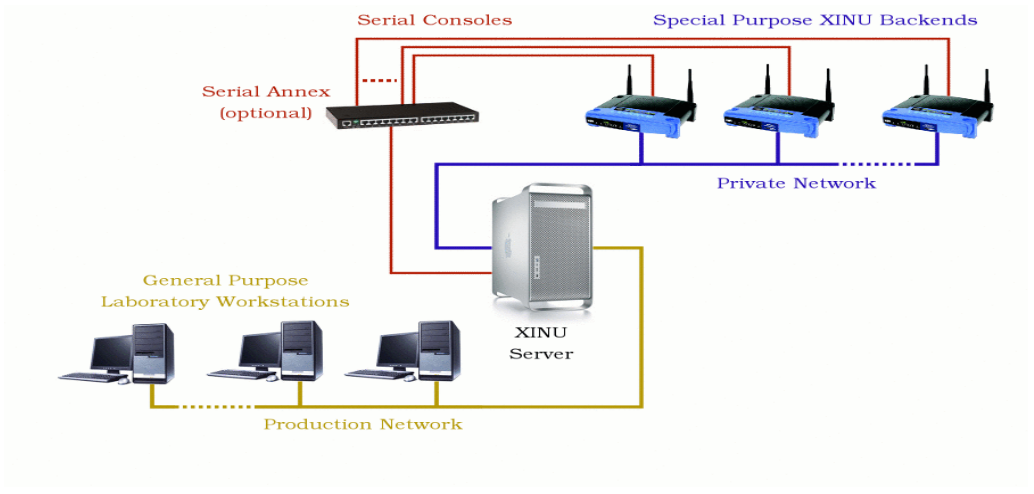

Figure 2 Expanded NEXOS environment with a central server

Figure 2 outlines the typical layout first used at Purdue and later nicknamed the *warzone* by the TinkerNet Project. A cadre of dedicated target machines resides on the private network (warzone) where students may load and run experimental kernels on demand. These dedicated, backend machines load a new O/S image from the central server across the network each time they reboot, allowing students to refresh the image frequently as they build their system.

3. What to do?

Section 3.1, 3.2, 3.3 describe stand-alone operations. While you can try this for your own interest we have prepared the lab for you, by making the modifications given below. These are given for completeness-sake.

3.1 Prepare Hardware

1. Study the various components of the environment discussed above.
2. Modify the Linksys WRT54GL hardware as follows: Obtain the parts as given in the list. Parts list:

Quantity	Part Name	Details	Part / Model Number
1	LinkSys WRT54GL Router	802.11b/g wireless broadband router	Linksys WRT54GL
1	Ribbon cable	28 AWG, 10 conductor, 25'	Jameco 643508CM
1	IDC socket connector	0.1”, 10 conductor	Jameco 32491CM
1	IDC shrouded header	0.1”, 10 conductor	Jameco 67811CM
1	RS232 Transmitter/Receiver	IC 2DVR/2RCVR RS232 5V 20-DIP	DigiKey MAX233CPP-ND
1	DB9 Female	22AWG,SOLDER CUP	Jameco 15771CM
1	DB9 Male	22AWG,SOLDER CUP	Jameco 15747CM

3. Open the router.
4. Attach the serial header.
5. Wire serial header to MAX233A RS232 receiver/transmitter.
6. Wire the DB9 connectors to MAX233A.

Make sure you solder the socket before mounting the chip itself.

3.2 Prepare software

1. Download and cross-compile the embedded XINU code.
2. Download TFTP for loading compiled XINU into the router via CFE.
3. Study the XINU code in the subdirectories.
4. Connect the serial port UART0 to the host (laptop/computer) with embedded XINU code and load it.
5. Configure the serial communication software.
6. Power-up the router.
7. Access the CFE and upload the embedded XINU.

3.3 Test the setup

Build and deploy XINU and examine the commands available. Write a simple test program "Hello World" and test the setup.

3.4 Study Embedded XINU files

Download the Embedded XINU bundle (tarball) from http://xinu.mscs.mu.edu and untar it. 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, gpio.h, interrupt.h, shell.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.**

3.5 Write simple shell commands (15 points)

The XINU shell is a subsystem designed as an interface for interacting with the operating system. The shell relies on the TTY driver to receive user input and provide output to the user. When XINU starts, a shell process is spawned for each serial port (or TTY). TTY operates on the UART driver. Study the steps given in http://xinu.mscs.mu.edu/Shell to understand how to create a shell command. Add this shell command:

reverse *string* that reverse the *string* input and prints it out.

> **reverse unix**
> **xinu**
> **reverse emit**
> **time**

3.6 Study and understand some more commands:

- **system (system operations): initialize.c, main.c**

The following code from initialize.c will create the main process and shell 0 (TTY0/Console):

```
open(CONSOLE, SERIAL0);
 ready(create((void *)main, INITSTK, INITPRIO, "MAIN", 2, 0, NULL), RESCHED_NO);
```

The following code in main (main.c) initializes the Shell 1 to be TTY1/SERIAL1:

```
 process main(int argc, char **argv)
{
 /* Associate TTY1 with second serial port. */
 open(TTY1, SERIAL1);
 enable();

 /* Launch one shell process for each TTY device. */
 ready(create((void *)shell, INITSTK, INITPRIO, "SHELL0", 1, CONSOLE),RESCHED_NO);
 ready(create((void *)shell, INITSTK, INITPRIO, "SHELL1", 1, TTY1), RESCHED_NO);
```

Thus there is a root process and main process and shell0 and shell1 (corresponding to uart0 and uart1 of the UART).

- **Shell directory: Shell.c file**

Read the data structure associated with a shell process and how a line buffer is handled, tokenized etc.

```
process shell(ushort descrp)
{
	char	buf[SHELL_BUFLEN];		/* line input buffer	*/
	ushort	buflen;			/* length of line input	*/
```

Do you understand how to create a new process now by looking at the above process definitions and process creations?

- **all the files in tty and uart directories.**

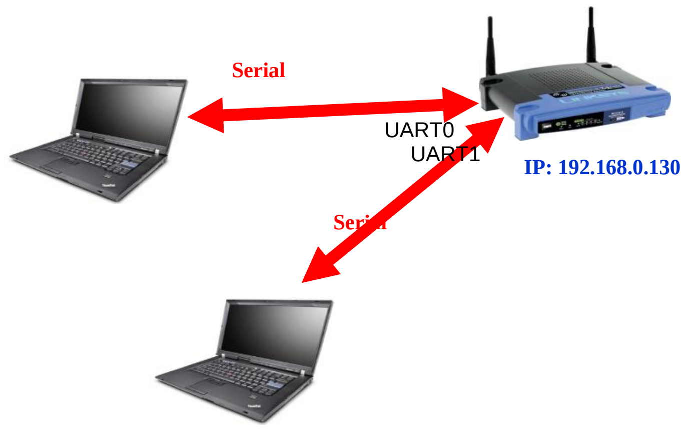

Fig 3: Network to enable serial network with Embedded XINU /WRT54GL platform

3.7 Periodic real-time Tasks Scheduling (30 points)

In this component of the project you will use **Xinu threads** to implement n periodic tasks.
For the task set given below design a cyclic schedule: (i) determine the hyper-period (ii) determine frame size, (iii) provide a timing chart and (iv) a cyclic (executive) schedule. ti is the task#, ri is the arrival time, ei is the execution time, pi is the period and Di is the relative deadline.

ti	ri	ei	pi	Di
t1	0	2	8	6
t2	1	2	6	6
t3	0	4	12	10

Based on the schedule write a Xinu program with (i) built-in schedule and (ii) built-in schedule to carry our these periodic tasks. **Each task will be represented by a xinu process (thread).** We discussed this in class and a solution is available in your text and in the demoes directory. You have to port it to XINU environment.

3.8 Write a chat program (30 points)

a. Add a chat program with login feature that will prefix messages with the originator's name and print it on the device (lap top).

b. Extend the chat program to allow multiple user (more than 2) and allow chatters to join and leave. In this case there is a super user admin who can kick out chatters with problems. Assume that each chatter (person) is inherently a "problem chatter" with certain probability!

3.9 Priority-driven Cyclic Executive and Priority Inversion (15 points)

For this exercise you are required to simulate the priority driven realtime scheduling like the one on Mars Rover [7]. Show using Xinu threads the occurrence of priority inversion. Then use priority inheritance to address the priority inversion problem. You may have to use the newer version of Xinu to get the various priorities working.

4. References

[1] WRT54GL: http://www.linuxelectrons.com/features/howto/consolidated-hacking-guide-linksys-wrt54gl/print
[2] EXINU: http://xinu.mscs.mu.edu
[3] WRT4GL : http://www.wi-fiplanet.com/tutorials/article.php/3562391
[4] EXINU memory: http://xinu.mscs.mu.edu/HOWTO:Deploy_Xinu
[5] CFE: http://xinu.mscs.mu.edu/CFE
[6] TFTP: http://www.tftp-server.com/tftp_server_overview.html
[7] What really Happened on Mars?, An email/article by Mike Jones,Dec 1997.
http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/mars_pathfinder.html
[8] P. Asadoorian, and L. Pesce, WRT54G Ultimate Hacking Guide, edited by Raul Siles, Syngress, 2007. https://we.riseup.net/assets/48812/Linksys%20WRT54G%20Ultimate%20Hacking.pdf