A pool of Raspberry Pi's running Xinu for college computer science curriculum

Jason Arnold, Casey O'hare, Omokolade Hunpatin, Luke Mivshek

Marquette University, Wisconsin

Introduction

The Embedded Xinu project is being conducted by both graduate and undergraduate students in the Systems Laboratory in the Math, Statistics, and Computer Science department of Marquette University in Milwaukee, Wisconsin [1]. The existing system, which is being used for Operating Systems, Hardware Systems, Embedded Systems and Networks courses offered at Marquette University, uses a pool of Linksys WRT54GL routers to run custom student made Xinu kernel images. The reason why we chose to convert the existing system from a pool of Linksys WRT54GL routers to a pool of Raspberry Pi's was because Raspberry Pis are:

- 15 dollars cheaper than the routers
- more open than the routers
- more modern than the routers
- smaller than the routers

Figure 1: A picture of a Raspberry Pi model B.

MIPS Vs. ARM Architecture

The Linksys WRT54GL routers use MIPS architecture processors compared to the Raspberry Pi, which uses ARM architecture processors. Biggers (2013)[3] already ported the Xinu operating system for the Raspberry Pis, so they rewrote the supporting Xinu kernel MIPS assembly code to ARM assembly. Unfortunately, the courses at Marquette, which used the Embedded Xinu project, has all of its assignments, tests and quizzes use MIPS assembly language. Therefore, we had to convert the MIPS assembly assignments to ARM assembly to ensure this was viable for the fall 2015 semester. As figure 2 shows, MIPS does have more general purpose registers than ARM. However, we found that the students benefit greatly from the switch from MIPS assembly to ARM assemly in that they can reduce their code for a lot of assignments, such as context switching processes. Additionally, students benefit from using Arm assembly in that when specifying a stack pointer offset to load or store operation, one can use a register in ARM. This allows a student to write assembly programs a lot easier.

MIPS general purpose registers

Register Name	"regdef.h" Naming Convention	Possible Usages
$0	zero	wired to zero value
$1	$at	reserved for the assembler
$2	$v0	return values
$3	$v1	
$4	$a0	arguments
$5	$a1	
$6	$a2	
$7	$a3	
$8	$t0	temporary
$9	$t1	
$10	$t2	
$11	$t3	
$12	$t4	
$13	$t5	
$14	$t6	
$15	$t7	
$16	$s0	saved
$17	$s1	
$18	$s2	
$19	$s3	
$20	$s4	
$21	$s5	
$22	$s6	
$23	$s7	
$24	$t8	temporary
$25	$t9	
$26	$k0	reserved for the OS kernel
$27	$k1	
$28	$gp	reserved for the global pointer
$29	$sp	reserved for the stack pointer
$30	$s8, $fp	saved, frame pointer
$31	$ra	linker register (return address)

ARM general purpose registers

Register Name	Usage
R0	general purpose (scratch)
R1	
R2	
R3	
R4	general purpose
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	reserved for the stack pointer
R14	linker register
R15	reserved for the program counter

Figure 2: MIPS Architecture general purpose registers and ARM Architecture general purpose registers side by side

MIPS vs. ARM Assembly[4]

Mailbox property interface

When we first started trying to set up the pool of Raspberry Pi's, we initially wanted to test communicating with the Raspberry Pi's over the network. We found that the port of Xinu for the Raspberry Pi couldn't find the hardware MAC address for the Raspberry Pi and would generate a random hardware MAC address every time the Raspberry Pi booted up. This was a problem because to communicate with a Raspberry Pi over the network, it needs a static hardware MAC address that needs to be known in order for us to designate it an IP address.

Figure 3: The shell command pistat

Solution:

We found out about the Mailbox property interface, which is the primary means of communication between the ARM and the VideoCore firmware running on the GPU [5]. Using the mailbox property interface we created a Xinu shell command called pistat that reads a Raspberry Pi's hardware information. Additionally, we created functions that takes a parameter of what hardware information a user wants then returns that hardware information.

The general procedure to read from a mailbox:

1. Read the status register until the empty flag is not set
2. Read data from the read register
3. If the lower four bits do not match the channel number desired then repeat from 1
4. The upper 28 bits are the returned data

[6]

Acknowledgments

The students on the building a pool of XinuPi's team were supported by the National Science Foundation, grants CNS-1339392, and ACI 1461264. The porting of the Xinu Operating system from the Linksys Routers to the Raspberry Pi and the shell command that allows one to receive a kernel's binary through tftp and boot itself using that kernel's binary in RAM is thanks to Eric Biggers, Tyler Much, and Farzeen Harunani [3].

References

[1] http://xinu.mscs.mu.edu/Main_Page

[2] https://www.raspberrypi.org/help/what-is-a-raspberry-pi

[3] Biggers, Eric, et al. "XinuPi: Porting a Lightweight Educational Operating System to the Raspberry Pi." 2013 Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education. 2013.

[4] http://www.ece.gatech.edu/academic/courses/ece2035/readings/embedded/MIPSvsARM.pdf

[5] https://github.com/raspberrypi/firmware/wiki

[6] https://github.com/raspberrypi/firmware/wiki/Accessing-mailboxes

[7] https://www.raspberrypi.org/blog/price-cut-raspberry-pi-model-b-now-only-25/