ECE 598 – Advanced Operating Systems
Lecture 9

Vince Weaver

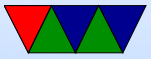
http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

16 February 2016

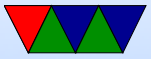
Announcements

- Homework #4 was posted
- Homework #3 was graded

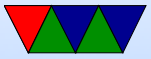


HW#3 Overview

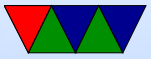
- be sure your code compiles!
- printk: instead of /10, print remainder plus '0' instead /16 (which converts to shift) and two cases. 0-9 same as before, but A-F (you can just add 'A'-10 which I think is 55)
  technically upper vs lowecase %X vs %x
- Be sure print hardware info (r1)
- Why serial port?



- Why parity?
- Why no parity?
- inline asm
- Why no strtok?
- Problems with OSX?



Things added for HW#4



ATAGS

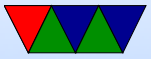
- List of variables passed by bootloader. A standard. See:

  `http://www.simtec.co.uk/products/SWLINUX/files/booting_article.html`

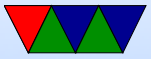
- We mostly care about getting memory size and machine type.

- Size, Tag-type, additional

- Located traditionally at 0x100 but you should really check r3 for addr.

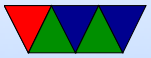


- Starts with ATAG_CORE
- Ends with ATAG_NONE
- We wants ATAG_MEM and maybe ATAG_CMDLINE on Raspberry Pi.
- Format is SIZE, TYPE, DATA0 ... DATAN. Then repeat.



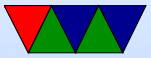
include and quotes

- What is the difference between `#include <string.h>` and `#include "string.h"`
- The first looks at the system includes
- The second looks in your local directory (or what you specify with `-I` on the command line)



string manipulation

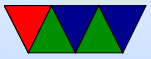
- Most C-based OSes quickly obtain string manipulation functions
- strncmp(), strlen(), strncpy(), memcpy(), memset(), memcpy()
- What's the different between strncpy and memcpy?
- How optimized do these routines need to be?
- memcpy() is often short blast of C



```
for(i=0;i<n;i++) { *d=*s; d++; s++;}
```

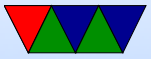
but it can be optimized to death.

- Should I mention memmove difference? Why it's there, hazzard when you don't use it right? (memmove the areas can overlap) (what happens if you copy backwards)



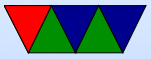
no more \r

- I've modified uart_write() so you no longer need to do \r.



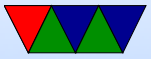
writing a shell

- What is a shell, or monitor routine?
- How can you parse a command line?
- Read values into a buffer. When enter pressed, check for a command. strcmp()? By hand? strtok() if fancy?
- Do whatever the command indicates, then reset buffer pointer.
- Print an error if unknown command.



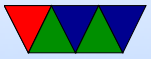
LED routines

- I added LED routines in led.c along with gpio.c
- This abstracts the code away, so it shuold work on any kind of Pi transparently (though very slightly slower than direct coding it)
- Good for you, but also makes grading easier for me.



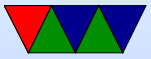
Interrupt Roundup

Any questions on interrupts?



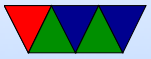
Interrupts Schemes

- More info on nested interrupts
- More info on interrupt priority



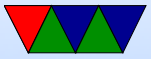
Interrupts on Linux

- Can look in `/proc/interrupts`
- Latency matters. Old days had problems where you'd lose serial interrupts (small FIFOs) if your disk drive took too long, etc.
- Cannot do anything that might block in an interrupt. Can you do I/O? Can you do a printk?
- Top Half / Bottom Half
  Have interrupt routine be bare minimum short. ACK



interrupt, handle super pressing thing (copy data out of FIFO) Then tell the kernel to handle the rest later.
So you might have a tasklet/kernel thread that runs occasionally (and is fully interruptable) that will do the rest.
For example, network packet comes in, important to read the packet and ACK interrupt. Put it in queue, then later the code that does longer latency stuff (decodes packet, does ethernet or TCP/IP stuff, then finally copies the data to the code waiting)

- Timer interrupt. How often? 100Hz originally. Up to



1000Hz (why?) now configurable, often at 250Hz.