ECE 471 – Embedded Systems
Lecture 13

Vince Weaver

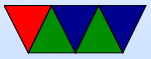
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

2 October 2024

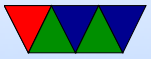
Announcements

- Don't forget homework #4

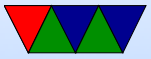


Homework #3 Review – Exit in Assembly

- Exit – value is an integer which goes into r0 (x0 on 64-bit)
- Note it is an integer, not ASCII
- Be sure to comment your code and fix any wrong comments
- Note: ABI says value going into r0 is first argument, not the return value
- Only one line changed needed, some people misunderstood

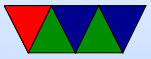


- 95% of students seem to be on 64-bit OS



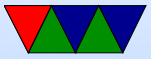
Homework 3 – ARM32 vs THUMB2

- be sure to specify base!
- ARM32 – `0x1048C - 0x1041c = 0x70 = 112 bytes`
- Thumb2 – `0x1046C - 0x10414 = 0x58 = 88 bytes`
  Note on new compiler programs might be loaded at 0x500 instead
- Note it's bytes not bits. Also no need to divide by 4. Also each hex digit is a nibble
- Differences?
  - Thumb2 some instructions are 16-bit rather than 32-



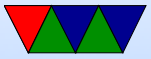
bit

- Thumb2 different instructions (like movt/movw)
- Thumb2 short instructions can be 2-argument



Homework 3 – Code Density

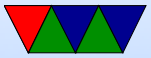
- You need to run `strip` on this to see it. Why?
  Debug info, including extra thumb debug as well as the longer filename.
- You can use `readelf -a` and `readelf -s` to see the space the various segments take up.
  Look at executables, *not* the C source code.



Homework 3 – Code Sizes

arch	unstripped	stripped
C arm32	9,896	5556
C thumb2	9,900	5556
asm arm32	1,308	536
C static	573,096	485,648
C ARM64	70,480	67,600

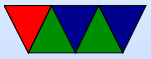
- You would think THUMB2 would be much smaller, but the assembler makes some poor decisions about wide/narrow instructions.



- Reference my LL work
- C code is larger, but also remember to include the C library:

```
 ls -lart /lib/arm-linux-gnueabihf/libc-2.31.so
-rwxr-xr-x 1 root root 1321488 Sep  8 09:17 /lib/arm-linux-gnueabihf/libc-2.31.so
```

- There are embedded C libraries, musl, newlib, uclibc, which are much smaller and often used in embedded systems.
- Smallest possible executable? I have written 128 Byte ones for competitions but you have to do sketchy things to the ELF file to be that small

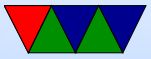


Homework 3 – gdb

- crashes!
- have to use awful gdb interface
- line 9 is the crash
- the assembly is

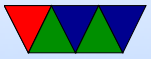
  ```
  ldrb r3,[r3]    // or ldrb w0,[x0]
  ```

  load byte from the 32-bit address pointed to by r3, store the resulting zero-extended byte into r3 (replacing the old value)
- if you look at src code or info assem you can see it's



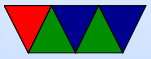
dereferencing (following) a NULL (uninitialized) pointer, which is always a segfault on Linux

- Note in this particular case it's not an "off the end of the array" issue, but rather the array doesn't exist at all problem
- Don't confuse NUL terminated strings with invalid NULL pointers



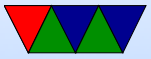
Homework 3 – How would you convert to Hex instead

- How would you convert print_number to hexadecimal?
- Is it easier to divide by 16 than 10? Especially w/o a divide instruction?
- Yes, shift and masks. Trick part is to special case 10 to 15 to be A to F
- If read with scanf(), do you handle negative numbers? Do you handle floating point numbers? Characters? Hex numbers?

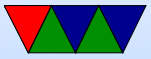


Homework 3 – Linux Tools – cal 9 1752

- Debian Linux dropped "cal" from the default install, it's now in the "ncal" package
- cal missing days
- Julian to Gregorian calendar.
- People sad who paid weekly but paid rent monthly.
- George Washington's b-day / Hunt for Red October
- Beware believing any page you google. Some urban legends / joke sites about this. If it were some sort of programmer bug it would have been fixed years ago.



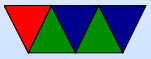
Briefly reviewed Virtual Memory from Last Time



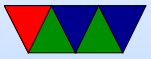
Coding Directly for the Hardware

One way of developing embedded systems is coding to the raw hardware, as you did with the STM Discovery Boards in ECE271.

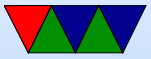
- Compile code
- Prepare for upload (hexbin?)
- Upload into FLASH
- Boots to offset



- Setup, flat memory (usually), stack at top, code near bottom, IRQ vectors
- Handle Interrupts
- Must do I/O directly (no drivers)
  Although if lucky, can find existing code.
- **Code is specific to the hardware you are on**

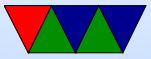


Instead, one can use an Operating System

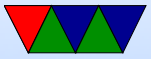


Why Use an Operating System?

- Provides Layers of Abstraction
  - Abstract hardware: hide hardware differences. same hardware interface for classes of hardware (things like video cameras, disks, keyboards, etc) despite differing implementation details
  - Abstract software: with VM get linear address space, same system calls on all systems
- Other benefits:
  - Multi-tasking / Multi-user

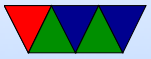


- Security, permissions (Linus dial out onto /dev/hda)
- Common code in kernel and libraries, no need to re-invent
- Handle complex low-level tasks (interrupts, DMA, task-switching)



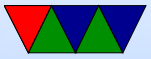
Downsides of Operating System?

- Overhead / Abstraction has a cost
  - Higher overhead (speed)
  - Higher overhead (memory)
  - Unknown timing (Real Time)
- Security
  - Larger code base can provide larger attack surface



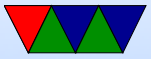
Other Aspects of Operating Systems

- What about other things?
  - Easy to code for? Provide examples
  - Nice GUI interface? Sometimes



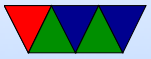
What's included with an OS

- kernel / drivers (syscall barrier) – Linux definition
- also system libraries – Solaris definition
- low-level utils / software / GUI – Windows definition
  Web Browser included?
- Linux usually makes distinction between the OS Kernel and distribution. OSX/Windows usually doesn't.



Bypassing Linux to hit hardware directly

- On Raspberry Pi Linux does not necessarily support all possible low-level hardware
- For example, until recently you couldn't set advanced features of the GPIOs like pullups
- Also OS adds overhead, many syscalls to just turn on a GPIO line which bare-metal might be a single store instruction
- People have written code that will poke the relevant bits directly.

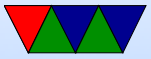


Bypassing Linux for speed

http://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/

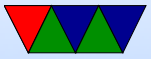
Trying to generate fastest GPIO square wave.

shell	gpio util	40Hz
shell	sysfs	2.8kHz
Python	WiringPi	28kHz
Python	RPi.GPIO	70kHz
C	sysfs (vmw)	400kHz
C	WiringPi	4.6MHz
C	libbcm2835	5.4MHz
C	Rpi Foundation “Native”	22MHz



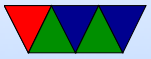
Operating Systems Types

- Monolithic kernel – everything in one big address space. Something goes wrong, lose it all. Faster
- Microkernel – separate parts that communicate by message passing. can restart independently. Slower.
- Microkernels were supposed to take over the world. Didn't happen. (GNU Hurd?)
- Famous Torvalds (Linux) vs Tannenbaum (Minix) flamewar



Common Desktop/Server Operating Systems

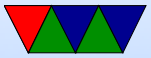
- UNIX derived
  - Linux (clone imlpemented from scratch)
  - FreeBSD / NetBSD / OpenBSD
  - MacOS (FreeBSD/Nextstep heritage)
  - Legacy (Irix/Solaris/AIX/etc.)
- Windows
- Obscure (BeOS/Haiku)



Embedded Operating Systems

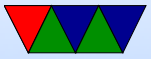
- Cellphone/Tablet
  - Android (Linux)
  - ChromeOS (Linux)
  - Apple iOS
  - Microsoft (WinCE/Mobile/Phone/RT/S/IoT (all these have been discontinued))

    In theory can install Windows 11 on a Raspberry Pi
- Networking
  - OpenWRT (Linux)



  - Cisco iOS
- Real Time OS
  - VXworks – realtime OS, used on many space probes
  - QNX – realtime microkernel UNIX-like OS, owned by Blackberry now
  - ThreadX – found in Pi GPU, Microsoft owns now?

    `https://www.theregister.com/2023/11/28/microsoft_opens_sources_threadx/`
  - FreeRTOS



Embedded Linux Distributions

This list is horribly out of date.

- linaro – consortium that work on ARM software
- openwrt – initially designed for wireless routers
- yocto – Linux Foundation sponsored embedded distro
- maemo – embedded distro originally by Nokia (obsolete)
- MeeGo – continuation of maemo, also obsolete
- Tizen – Follow up on MeeGo, by Samsung and Intel
- Ängstrom – Merger of various projects