

ECE 471 – Embedded Systems

Lecture 12

Vince Weaver

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

`vincent.weaver@maine.edu`

30 September 2026

Announcements

- Don't forget homework #4
Please use the interface described in class to access the GPIOs
- HW#2 grades finally posted. Don't forget to comment code!

HW2 Review – General

- Most people seem to be accessing the Pi OK
- Put your name in the README
- Testing. How can you test? `wc -l`

HW2 Review – C Code

- Comment your code!
- Be sure if it says print 12 lines that it does.
- Watch for off-by-one errors
- Colors seem not to be a problem.
- Also watch out for compiler warnings! (Though each compiler version might have different warnings)
- **Error handling! especially for command line parsing**
Crashing is bad, especially in embedded systems

HW2 Review – Questions

- more info on ls. Looking for man. “info” or `ls --help`
- `ls -a` shows hidden files. Hidden files on UNIX
- Why use C? close to hardware, easier than assembly, etc.
careful with “faster”. It often is, but give reasoning and what you are comparing (faster than python? asm?)

HW2 Review – Connecting to the Pi

- People connecting OK?
- Originally point of Pi was cheap board and you possibly already had SD card, USB-keyboard (from desktop), USB-micro charger, and HDMI cable floating around. Modern Pis more difficult, mostly the micro-HDMI cable which I don't think anyone has.
- Network? ssh? VNC? keyboard/screen? Bluetooth?

Operating Systems

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.

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 Operating System Benefits

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

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.

Language	Method	Max Toggle
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 implemented from scratch)
 - FreeBSD / NetBSD / OpenBSD
 - MacOS (FreeBSD/Nextstep heritage)
 - Legacy (Irix/Solaris/AIX/etc.)
- WindowsNT (NT/2000/XP/Vista/8/10/11)
- CP/M, DOS (Windows 3.1, 95, 98)
- Obscure: BeOS/Haiku, hurd, RiscOS

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

More Embedded Operating Systems

- 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/](https://www.theregister.com/2023/11/28/microsoft_opens_sources_threadx/)
 - FreeRTOS
 - Minix – ended up in lots of Intel machines

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
- Ångstrom – Merger of various projects