F28HS Hardware-Software Interface: Systems Programming

Hans-Wolfgang Loidl

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh

Semester 2 2016/17

[0]No proprietary software has been used in producing these slides

Outline

1. Lecture 1: Introduction to Systems Programming
2. Lecture 2: Systems Programming with the Raspberry Pi
3. Lecture 3: Memory Hierarchy
   - Memory Hierarchy
     - Principles of Caches
4. Lecture 4: Programming external devices
   - Basics of device-level programming
5. Lecture 5: Exceptional Control Flow
6. Lecture 6: Computer Architecture
   - Processor Architectures Overview
     - Pipelining
7. Lecture 7: Code Security: Buffer Overflow Attacks
8. Lecture 8: Interrupt Handling
9. Lecture 9: Miscellaneous Topics
10. Lecture 10: Revision

Lecture 9. Miscellaneous Topics

Bare-metal programming

- **Bare-metal programming** means “programming directly on the hardware”, i.e. on a system that doesn’t run an operating system.
- This is the most common scenario for embedded systems programming.
- In this course we used Raspbian on the RPi2 mainly for **convenience** (tool support etc)
- Embedded systems in industry usage are often too small to run any OS
- For time-critical operations you don’t want an OS because in order to meet **real-time** constraints.

What's different?

A lot:

- You have to control the boot process yourself
- You have to manage all aspects of the hardware directly:
  - memory (no virtual memory!)
  - external devices
- You need to produce stand-alone executables, i.e. no dynamically linked libraries
- You typically need to cross-compile your code

What are the advantages?

- You have direct control over the hardware:
  - For our LED etc examples, you **don't need** `mmap` to access the devices, rather you directly write to the hardware registers.
  - You can access aspects of the hardware that might not be accessible otherwise.
- Better suited for real-time constraints: no OS overhead, predictable performance
- Very small code size of the entire application
- Typically lower energy consumption

How does the application code differ?

Looking at our example code from the course

- No `mmap` is needed to access the GPIO pins
- You can't use external libraries: everything must be part of the application
- This means that in general you need to write your own device drivers for external devices such as a monitor
- The code typically needs to be cross-compiled, i.e. the machine that you are **compiling on** is different from the machine that you are **compiling for**.

And of course there are a lot of differences in terms of usability.

Further Reading & Deeper Hacking

- *"Embedded Linux"*, by Jürgen Quade (Textbook on embedded systems programming, using a bare-metal approach)
- Baking Pi, by Alex Chadwick (a course on bare-metal programming on the Rasbperry Pi at Cambridge University (only for RPi1))
- Valvers: Bare Metal Programming in C

Rust: an alternative systems programming language

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

[0] Rust

Rust Features

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

Internet of Things

- The amount of processors used in all kinds of settings is increasing rapidly.
- Examples are “smart homes” with configurable/programmable devices such as smart TVs etc
- These typically use small, embedded devices
- These devices want to exchange data, e.g. to monitor the environment and react to changes
- Therefore, these systems are inter-connected, building an **Internet of Things**
- These systems increasingly use a full operating system underneath
- Thus, a **RPi 2 running Raspbian is a good case study**

OS choices for the Internet of Things

- Rapbian, while useful as an interactive OS, comes with a lot of unnecessary packages if it should be used on one of these networked, embedded devices.
- Smaller, configurable Linux versions are often a better choice, e.g. Arch Linux (also available for RPi2).
- These reduce the resource consumption of the system, and improve maintainability.
- Several new[1] OS’s target this market: for example **MinocaOS**

[1] There are also several old OS’s that fit this characterisation: see Minix and RISC OS.

Main features of MinocaOS

- MinocaOS is a completely new OS, matching standard interfaces such as POSIX.
- MinocaOS is advertised as: `Modular`, `Lean`, `Flexible`
- MinocaOS supports RPi1 and RPi2/3 in 2 different images that can be downloaded
- There is no 64-bit support available yet[2]
- MinocaOS is also provided as a Quemu-based virtual machine, for experimentation on a laptop
- MinocaOS has a very small resource footprint, and works well even on older RPi1's
- MinocaOS has good hardware support and fairly good tool support

[2] See the slides at the end for a link on how to build your own 64-bit kernel on an RPi3

MinocaOS

Some notable features of MinocaOS are:

- Most command-line tools are based on GNU versions: `bash`, `ls`, `cat`, `chmod`, `nano` (use `--help` to get info)
- It uses package management similar to Debian-based systems (`opkg` as package manager; packages have extension `.ipkg`)
- The list of available packages and repos can be edited in `/var/opkg-lists/`
- **No** graphical user interface at the moment (not necessary for IoT context)

A Guided Tour is available on the MinocaOS web page.

[2] Material from Raspberry Pi Geek 04/2017

UBOS: easy configuration

- UBOS is a Linux distribution for easy management of several web services on an Rpi.
- Very flexible, being based on Arch Linux
- Features (as advertised):
  - With UBOS, web applications can be installed, and fully configured with a single command.
  - UBOS fully automates app management at virtual hosts
  - UBOS pre-installs and pre-configures networking and other infrastructure.
  - Systems that have two Ethernet interfaces can be turned into a home router/gateway with a single command.
  - UBOS can backup or restore all, or any subset of installed applications on a device
  - UBOS uses a rolling-release development model
  - UBOS itself is all free/libre and open software.

3

[3] Material from Raspberry Pi Geek 04/2017

Compiling an 64-bit kernel for RPi3

A detailed discussion on how to build a 64-bit kernel on a Rasberry Pi 3 is given in the Raspberry Pi Geek 04/2017.
A pre-pared 64-bit image for the RasPi 3 is here