ECE 785 Final Exam Study Guide (Spring 2021)

Open book, open notes, open computer. Assume that C code is compiled for the ARM Cortex-A72 following the ARM Architecture procedure calling standard (AAPCS).

Building Embedded Systems

- Programming languages
  - Performance implications of shell scripts, interpreted languages, compiled languages
  - Treating I/O devices as files
- GPIO interfacing
  - GPIO hardware peripheral concepts
  - Methods: sysfs (and I/O stream), memory-mapped I/O with mmap and dev/mem, gpiod, kernel thread, other libraries
    - Concepts
    - Performance
  - Trade-offs
- Asynchronous inputs
  - Methods to wait for inputs
    - Without O/S: busy-waiting
    - Using O/S: (p)select, (p)poll, epoll_wait,
  - Program structure
    - Single thread, threaded callback, signal handler, interrupt
    - Trade-offs
- Loadable kernel modules
  - Commands for basic use
    - insmod, lsmod, rmmod
  - Actions and behavior
    - Loading and parameters
    - Unloading
    - IRQ Handlers
    - Accessing attributes
    - Threads
  - Code Structures and Data Structures

Examining Object Code

- Platform
  - ARMv7-A Instruction Set Architecture
    - Integer instructions
    - Floating point instructions
    - Advanced SIMD instructions
  - Cortex-A72 processor features
    - Instruction processing core
    - Memory hierarchy
  - Raspberry Pi 4
- Given a source code listing
  - Create a function call graph
- Given object code (an assembly code listing),
  - Identify which machine instructions implement specific source code operations
  - Identify which registers or memory locations hold specific variables
  - Create a function call graph
  - Create a control flow graph

Analyzing and Optimizing Speed

- Profiling with PC sampling
  - Concepts
  - How to use perf for profiling
  - How to interpret perf's output
  - Limitations to profiling with PC sampling
- Given source code,
  - Identify operations which will dominate execution time
  - Propose and implement optimizations,
    - Reduce control flow hazards
    - Reduce impact of memory load penalty
    - Parallelize code to use Neon SIMD Unit
    - Apply polynomial approximations
    - Apply other optimizations
  - Estimate speed-up and development time costs for optimizations, rank optimizations based on estimates
- Given object code,
  - Identify operations which will dominate execution time
  - Propose optimizations
- Given source and object code,
  - Identify which optimizations actually were performed by the compiler
  - Identify code which was not optimized by the compiler. Possibly modify the source code to help the compiler optimize it.