CS510 Languages and Low Level Programming: Portfolio submission #2, Topic 2

Due on April 15, 2016 at 11:59pm

Mark P. Jones Spring 2016

Konstantin Macarenco

Attempted Topic:
Discuss common challenges in low-level systems software development, including debugging in a bare metal environment.

Narrative

There are many areas where low-level systems software development is used, for many different reasons, like OS performance optimization, removing middle layer software due to strict hardware requirements (like embedded systems), necessity of a small footprint etc.

Programming in low level mode proves to be difficult and error prone. It requires high attention to small details and good knowledge of the used hardware. There is very little, if none, abstraction that most of the programmers normally rely on, most of the OS facilities and API are not available. Additional challenge is the lack of trivial debugging tools (not talking about debugger), sometimes simple methods, like use of printf(), are out of consideration. For example a common debugging technique in low level applications for Raspberry PI, is to blink attached LED light, which is similar to POST beep codes, which BTW according to specs are not always guaranteed. There is no easy way to go through program's logic, inspect memory and variables values. Deep understanding of the used hardware, such as ABI, CPU Registers layout, Boot Process, Memory Management, Context Switching, Interrupts etc. is vital. Learning these technical details is a dull and meticulous task, for example Intel x86-32 documentation is more than 3000 pages long.

Some Low level systems debugging approaches.

1. Run low bare metal application within instrumented virtual machine that emulate desired computer architecture, and provide similar functionality as a debugger. There are some issues with this approach: this type of "Debuggers" are usually costly (up to $10+k).

2. Another approach is to create code that will work equally under bare metal systems and under normal OS, with only recompilation required to transfer from one to another, however this is not always achievable, since low level environments are dramatically more sensitive to undefined behavior, especially uninitialized variables. This problem can be somewhat mitigated by using all possible levels of compiler optimization, and checks. After all, the products will still require thorough testing on the actual hardware.

3. QEMU + gdb. QEMU provides integration with gdb, it acts as a remote gdb server. Kernel needs to be compiled with debug symbols ("-g" flag). One downside is that debugging symbols will significantly increase kernel's size.

Modern high level languages achieve better productivity and quality, over low level languages like C and assembly. They considered to be "safe" with much better memory management, garbage collection, and other feature that are not available for Low- Level systems programming. C and assembly are very powerful, but unforgiving to inexperienced programmers, they require lots of management and easy to make mistakes. The situation is exacerbated by inability to use Standard C-Library, since it heavily relies on OS features. In bare metal means that you start with very little external interfaces, no console, no file system etc. This problem is especially significant in environments without common debugging tools, like bare metal systems.