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 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. As mentioned before deep understanding of the used hardware, such as CPU Registers, Context Switching, Interrupts, MTU etc., is the essential part of such a development, without it even simple task can create many potential problems, learning these technical details is a dull and meticulous, for example Intel x86-32 documentation is more than 3000 pages long. Most modern high level languages that considered to be "safe", like Java, Python, C# are not usually available for Low-Level systems programming. Most often programs for bare metal, written in combination of C and assembly. These languages are very powerful, but unforgiving to unexperienced programmers. Most of the current security issues are created by their use. This problem is especially significant in environments without common debugging tools, like bare metal systems.

However there are several approaches, that try to solve some of the problems mentioned above.

1. Run low level systems 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 $6k), and these tools are typically available for most common computer architectures like x86. If you have something less common you are out of luck.
2. Another approach is to create code that will work equally under low level systems and under normal OS, with only recompilation required to transfer from one to another, however this is not always achievable.
3. Use visible state