COMP 250RTS Class Exercise: Imagining Compilers and Run-Time Systems

September 6, 2017

Today, the first day of class, we'll accomplish two things:

- Start to illuminate the design space of run-time systems
- Start to calibrate the class (and the instructor) on what we do and don't know

We'll do this by splitting into two groups and *imagining language implementations*. Each group will build out a table using this process:

A. *Think of a feature* or a group of features. Examples might include Smalltalk method dispatch, Scheme higher-order functions, ML-style polymorphism, Haskell type classes, Java exceptions, and so on. And plenty more where those came from.

   A feature may be a property of a *language* (first-class functions) or of its *implementation* (interactive read-eval-print loop).

   Each feature becomes a row of your table.

B. Imagine that the feature is implemented by a combination of compiler and run-time system. What does the compiler do? What must the run-time system do? Each answer goes in a column of your table.

Your goal is a broad, shallow overview of the things that run-time systems do—together with some understanding of where they come from. If you think of something a run-time system should do, but it's not obvious what language feature it goes with, put it up anyway.

Some fine points about targets and implementations:

- Imagine a sane, sensible target platform with (roughly) interchangeable machine registers and a nice, uniformly accessible address space. Think "server, desktop, and phone," or if you prefer, "AMD64 and ARM." Multicore targets are definitely in play.

  I'm ruling out weirdness like NUMA, segmented memory, the Playstation 3, GPU computation, and small devices like Arduino and ESP8266.

- Plenty of systems have dynamic compilers, but even in these systems, we distinguish the compiler from its run-time support.
- A system that compiles to bytecode (or VM code) still has a compiler.

Some fine points about language features:

- Let's stick to languages that have something like an evaluation model. No Prolog or SQL.
- For now at least, no lazy evaluation. It's a cross-cutting concern that influences every other part of a system.
- Bizarre stuff like MINT, TRAC, Tcl, and PHP is OK.
- Be sure to cover a reasonable part of the design space of type systems. What kind of run-time support is needed for a language with dynamic types? Static types? Polymorphic types?