Homework 6: Evaluate Zig as a language for secure camera-based HVAC control

Anh Mac (905-111-606)
University of California, Los Angeles

Abstract

We are looking into designing a software architecture for Haversack Inc.'s SecureHEAT system. This paper evaluates the strengths and weaknesses of Zig version 0.8.0 as a potential programming language to implement a HVAC system. The discussion will focus on the following factors associated with the language for its intended purpose, including security, ease of use, flexibility, generality, performance, and reliability. Overall, Zig poses as a very promising candidate for our intended application as a simpler and more lightweight version of its parent programming language C.

1. Introduction

The HVAC (Heating, Ventilation, and Air Conditioning) systems is intended to provide large customers with high and frequent usage a system to save money by controlling temperature in buildings more efficiently. They rely on a HEAT camera system that monitors the faces of human occupants to calculate facial temperatures and serve as a thermostat to control the buildings' temperatures. The cameras operate as an embedded system and complete most of its communications through a wireless network, so security is of high importance in designing its implementation. C and C++ have been traditionally used for such systems, but they have been criticized for its security concerns. Other priorities for the choice of programming language includes simplicity, ability to be a freestanding program, cannot rely on interpreters or garbage collectors, and compatibility with low-level hardware devices.

Zig is an imperative, general-purpose, statically type, compiled system programming language and toolchain made to maintain "robust, optimal, and reusable software." The four priorities of Zig are: pragmatic, optimal, safe, and readable. It has been largely considered as one of C's direct competitors, as it integrates with C standard library libc rather than depending on it. Zig is also a small and simple language with no hidden control flow, no hidden memory allocations, no preprocessor, and no macros. On the surface, Zig appears to be what Haversack would look for in its application, a simple and light language. We will further dive into discussions of more specific areas in regards to the language.

2. Security

Since many of Haversack Inc.'s customers manage buildings that house banks, the military, intelligence services, prisons, and other organizations that require high security, old systems that relied on C/C++ have had many concerns as these languages are more prone to outside attacks, leaking important and confidential information.

Zig uses manual memory management that makes it compatible with low latency servers, OS kernels, embedded devices, real-time software, and by any other languages that uses the C ABI. Different from traditional languages such as C and C++, Zig offers a new error handling method that uses `defer` and `errdefer` to make all resource management simple and easily verifiable. If an error occurs during runtime, `errdefer` immediately destroys the object and free the resource that was dynamically allocated. The language also offers other error handling method like the `unreachable` keyword to ensure that no errors will occur. Moreover, Zig also provides stack traces and error return traces on all targets that can capture a trace at any point in the program. This feature is also supported on freestanding programs.

3. Ease of Use

As mentioned before, Zig is designed to be simple and light. By making all control flow managed exclusively with language keywords and function calls, Zig promotes code maintenance and readability.

In addition to its simple interface, Zig is able to complete the C ABI Compatibility. Following the C ABI, an internal function can easily be exported and has its name show up in the symbol table verbatim. Recognizing the long-standing language's successes, like C, Zig also allows you to just use a library without creating a "wrapper" or "bindings" and provides the ability to parse .h files using the `@cImport` and `@cInclude` keywords.

Moreover, Zig is also compatible with other C programs as it produce simple .o files for linking or generate .h files for export. Since it follows the C ABI, users may easily link .o files together where one can be written in C and other in Zig. This forward compatibility would be suitable for our HVAC application, as it can help us transition the firm's old projects written in C to a simpler and more lightweight language like Zig. Different from C, Zig supports type inference by prepending functions with `fn` or variables with `var`. This makes our application more readable and easier to develop.

Additionally, Zig also offers extensive features like optional types, which can be used instead of null pointers, and support for concurrency with asynchronous operations. Since our system relies on network communication, this feature will be extremely useful for cameras to communicate with one another and to other hardware parts to control the temperature in the building as well.

4. Flexibility

Zig offers support for four different build modes, which can be mixed and matched all the way to scope granularity. Each of these build modes enable different levels of compile time performance, runtime performance, and undefined behavior handling. It also supports a wide range of targets and uses a "support tier" system to communicate the level of support for different targets.

Zig also has the concept of a debug build vs a release build with widely diverging characteristics for each of them based on metrics like Time Spent Compiling, Runtime Performance, and Undefined Behavior. Zig has also recently gained two more release modes which is Release Safe and Release Small. Having these options available to us would make the development process much easier for debugging and release schedule since there would always be features that suit our needs based on the state of our system.

5. Performance

In comparison with its ancestor language, Zig contains many more compile time optimizations in release mode to maximize runtime performance than C. By removing some inconsistencies involving calls and headers like `#define`, `#ifdef`, and `#include`, Zig's preprocessor ultimately performs much better than that of C. Zig provides support for conditional compilation through compilation variables available via `@import("builtin")`. Overall, Zig's runtime significantly improved compared to C as Zig contains a single compilation unit with all included libraries before optimizations are run. Thus, performance is improved by doing this instead of dynamically calling linked functions only when needed by the program.

6. Generality & Reliability

Improving from C weak type system, Zig drastically improve the reliability of the language with its stricter type checking and banning null pointers. First, Zig allows optional values. By using a `?` or `prepend` prefix to a function parameter, the programmer can indicate that the pointer is allowed to be null without the need of having to actually pass in a null value. It also provides a new method to handle error smarter by using error return codes. By using the `error` keyword, error codes form Error Sets or Error Unions, two primitive types that were introduced by Zig, that can be mixed, merged, and inferred. This feature in combination with the `defer` and `errdefer` feature as discussed before makes handling errors and doing proper cleanup much more drastically improved relative to C programming.

Having a reliable program is extremely important in our application as constant communication across networks can easily introduce uncaught errors if either types go unchecked or errors go unhandled. Thus, any potential errors will have the potential to shut down our entire system, so having a language that is stricter on these factors can indeed help increase reliability in the implementation of our systems.

7. Conclusion

After the in-depth discussion of its various offering of features and its characteristics, Zig seems to be a promising programming language candidate for our HVAC systems. Not only it is simple, light, and easy to use, Zig also ensure to maintain security and reliability with much improved performance compared to its parent C. It would be recommended to consider Zig as the language to program our systems as it also offers full compatibility with the C ABI that would allow us to slowly transition our older projects written in C to this newer and simpler language to ease the development process of future systems.

References

Zig Documentation

https://ziglang.org/learn/overview

Engheim, Erik. "Zig the Introduction." *Medium*, The Startup, 10 Nov. 2020, medium.com/swlh/zig-the-introduction-dcd173a86975.

Kelley, Andrew. "Introduction to the Zig Programming Language." *Introduction to the Zig Programming Language - Andrew Kelley*, 2 Feb. 2019, andrewkelley.me/post/intro-to-zig.html.