Executive Summary of Zig

Chuhua Sun

Abstract

This summary will investigate Zig, a programming language, on its effectiveness to support Haversack's Human Embodied Autonomous Thermostat (HEAT) system. This summary will examine strengths and weaknesses, along with problems of applications of Zig. We will focus on the technology's effects on security, ease of use, flexibility, generality, performance, and reliability.

Introduction

In this summary, we will focus on investigating a software architecture for Haversack Inc.'s SecureHEAT, a system intended to save money by controlling temperature more efficiently in buildings with the utilization of inexpensive cameras to implement a so-called Human Embodied Autonomous Thermostat (HEAT) system. HEAT system monitors the faces of human occupants, calculates their facial temperatures, and uses calculations to control the building's air temperatures. The cameras have no battery backup as they are attached to building power. They use a wireless network to connect to base stations at secure locations inside the building. The cameras utilize embedded CPUs with limited processing power and memory to do the bulk of calculations with the incoming data and control the building's HVAC units accordingly. Since SecureHEAT customers include high security-required organizations such as house banks, the military, intelligence services, and prisons, the designed system must have very high security to prevent penetration and leak of software in those cameras. Since Haversack has used a mixture of C and C++, which do not provide the required security, we will investigate Zig as an alternative language to program such systems in the past. Additionally, there are some more properties that the alternative language should support. First, the software must be clearly written and easy to audit. Second, it should be a freestanding program. Third, it should be as simple and stripped-down as possible. And it must be able to interface to low-level hardware devices. In this summary, we will evaluate Zig as a candidate for this application. Specifically, we will investigate the strengths, weaknesses, and potential problems of Zig along with its effects on security, ease of use, flexibility, generality, performance, and reliability.

Zig

Overview

Zig is a general-purpose programming language and toolchain for maintaining robust, optimal, and reusable software. It enables developers to focus on debugging with the application instead of programming language knowledge since it has no hidden control flow, no hidden memory allocations, and no hidden preprocessor or macros. It introduces a new method to metaprogramming based on compile-time code execution and lazy evaluation. Hence, it enables developers to call any function at compile-time, manipulate types as values without run-time overhead, and emulates the target architecture. It also allows developers to write fast and clear codes that enable all-type error handling.

Strength

Zig is a simple and small programming language as its designer decided not to make it an object-oriented language in favor of easiness. Zig always follows developers' instructions since it does

not have any hidden control flow or memory allocation, making it super easy for the developers to locate and solve any errors. With the design of various build modes, developers could debug Zig codes in an even simpler and more efficient way. Moreover, as the designer pays a lot of effort to Zig's security, it could be considered fairly safe for our system. While developers are allowed to import C code to Zig, it can fully compile C code. With the asyncio library similar to that of Python, Zig supports concurrency without real parallelism. Zig is designed as a system language, enable developers to easily create freestanding programs that support strong interaction with low-level hardware, making it a good option for our system.

Weakness and Potential Problem

The first weakness of Zig is its maintainability and readability. Even though Zig is a C-like language, static typing in Zig does not work in the same way like C, hence damaging the readability and maintainability of the codes if maintained by multiple developers who were not previously proficient in it. Typing issues also impact developers since the declarations of variables with type are designed in a very unpopular, complicated, and weird way, so it takes time and effort for developers to get used to it. Hence, since developers in Haversack Inc. were C/C++ developers, it might take their efforts to get familiar with Zig.

Security

Since our software is required to be as stripped down as possible, our programming language should be as low-level as possible and have no unreliable features such as garbage collectors. Zig has the same memory design as C since it only supports manual memory management, requiring developers to manually allocate and free dynamic memory. Notably, Zig has a built-in keyword called errdefer that automatically destroys an object with incorrectly allocated memory and helps developers manage resources reliably and efficiently. As Zig includes these security concerns, it meets our expectations for our system's alternative programming language.

Ease of Use

Generally, Zig is more readable and easier to write when compared to C since it has a significant focus on simplicity. As we mentioned previously, there is no hidden control or memory allocation in Zig. Developers can easily handle deferring and scope issues by using built-in keywords such as errdefer and defer. Moreover, comptime is a special feature provided by Zig to support compile-time computation. Zig can help developers to avoid run time errors caused by abusing null references by using optional types. Moreover, like Python, with the asyncio library's help, Zig could easily support concurrency applications with no real parallelism. Hence, Zig is straightforward to use, more unlikely to get an error, and simpler in concurrency applications such as interaction with cameras in our application.

Flexibility

Zig is very strong when we consider its flexibility since it supports four different build modes to support developers to create and debug codes more efficiently and concisely by alternating the degree of compile-time performance run-time performance, undefined behavior handling. Hence it provides notable flexibility for different developing phases of our application. At the construction stage, we could improve codes' productivity by working with the mode with high compile-time performance. Similarly, we could change to a mode that focuses on undefined

behavior handling when we are debugging our code. Such flexibility could make our development more efficient.

Generality

As we discussed previously, Zig enables snippets of C code to be imported. Moreover, the Zig compiler could fully compile C code, which provides an outstanding generality as a system programming language. In our application, developers can import various functions from C, which is extremely useful since our developers are very familiar with C. By ingeniously selecting functions from C and standard Zig library, developers can efficiently create a program that effectively interfaces with cameras in our system.

Performance

The preprocessor of Zig removes inconsistencies from various include processes in C, resulting in a faster preprocessor compared to C. As Zig is designed to be simple and concise with no garbage collector and virtual machines, it runs much faster than high-level programming languages. Since Zig supports different build modes that focus on respective optimizations, developers can tune the performance of our codes with high flexibility, producing a better run-time performance than C. While C utilized dynamically linked functions, each compilation unit in Zig is built before optimization, providing a stronger and automatic optimization. Thus, the performance of Zig is ideal for our system.

Reliability

Zig is strong in reliability as it provides full stack backtraces when we get some run time errors like file loading error, but C/C++, which generally produce no output. Therefore, developers could easily jump in and look at the top's message, which says the related error. Such very informatic stack backtraces are very powerful with a low-level system language. Since we are looking for a low-level system language to implement our application, Zig's notable reliability is valuable.

Conclusion

In general, Zig successfully meets all requirements of our system. Using Zig, we could build software that is cleanly written and easy to audit, a freestanding program with no separate operating system, also very simple and stripped-down, and able to interface to low-level hardware devices such as cameras. As a low-level system programming language, Zig provides excellent security, ease of use, flexibility, generality, performance, and reliability. Hence, Zig is a fairly well candidate for the alternative programming languages for our application.

References

UCLA CS131 Winter 2021 HW6: https://web.cs.ucla.edu/classes/winter21/cs131/hw/hw6.html

Zig Documentation: https://ziglang.org/learn/overview/

Understanding the Zig Programming Language: https://medium.com/swlh/zig-the-introduction-dcd173a86975

Introduction to the Zig Programming Language: https://andrewkelley.me/post/intro-to-zig.html