CS131 Homework 6 Executive Summary

Bryan Wong
Discussion 1A

Introduction

This executive summary is aimed towards designing a software architecture for Haversack Inc.'s SecureHEAT, a system that is intended to save money by controlling building temperatures in a way that is automatic and more efficient by using facial temperature sensing. This provides an "unobtrusive way to maximize comfort while using less energy", which could be especially beneficial during an era of COVID where individuals struggle to stay comfortable while wearing masks and other protective gear (University of Michigan 2020). SecureHEAT utilizes a Human Embodied Autonomous Thermostat (HEAT) system to monitor the faces of human occupants using inexpensive cameras, calculate occupants' facial temperatures, and use these calculations to control the building's air temperature settings. These cameras use embedded CPUs with limited processing power and are attached to building power, communicating to secure base stations using a wireless network. These base stations will do the bulk of the calculations using the incoming data and control the building's HVAC units accordingly. SecureHEAT customers will include high security organizations such as banks, military, intelligence services, and prisons. Therefore, the designed system must be highly secure as to prevent software in the cameras from being penetrated. The following are other important characteristics that our software must follow: 1. it must be cleanly written and easy to audit, 2. it should be a freestanding program, 3. it should be as simple and stripped-down as possible, and 4. it must be capable of interfacing to low-level hardware devices, such as camera or network interfaces. In this executive summary, we will compare and contrast three different programming languages— D, Zig, and Odin— and make a recommendation for which one best fits our system design. In addition, we will discuss the ethical ramifications of our choice of programming language.

Comparing D, Zig, and Odin

Overview

In this section, we will compare our three programming language candidates: D, Zig, and Odin. Each of these languages are all largely based on C/C++, with many shared features. Zig and Odin have been described as "direct competitors to C", albeit with specializations that help them find their particular niches. D was designed as a general-purpose systems programming language that attempts to make various improvements over C++. In this section, we will evaluate each of these three languages in the context of our application using a set of different criteria: security, ease of use, performance, flexibility, generality, and reliability.

Security

The first factor that we will discuss is security. Past systems that Haversack Inc. has created have used C and C++, which are vulnerable to attacks. Since the SecureHEAT system will be sold to high security customers, it is important that the choice of programming language allows us to prevent our camera software from being penetrated. If exposed, SecureHEAT cameras could leak confidential information to our customers' adversaries. This will also tie into ethical concerns surrounding the privacy of human occupants using our system, which we will discuss in a later section.

A first key tenet of our software design is that it should be as stripped down as possible, which means that our language of our choice should be as low-level as possible and without garbage collectors or virtual machines that customers may not trust. With the same philosophy to memory as C, both Zig and Odin use manual memory management. For these languages, the programmer must allocate and free dynamic memory themselves. Zig also

contains a handy `errdefer` keyword that makes resource management simple and easily verifiable. For instance, if an error occurs while dynamically allocating memory for an object in Zig, an `errdefer` can be made to automatically destroy the object. In contrast, D by default is garbage collected by pausing the current thread to recursively scan for unreachable allocated objects. However, D does have support for custom memory management; programmers can customize their programs to use manual memory management or implement a custom garbage collector of their own design. However, this requires extra work to accomplish and is not a first-class feature of the language.

Each language also has a variety of additional features that aim to improve security. D includes built-in testing and verification as a core part of the language. It also uses contract programming constructs `in` and `out` to ensure inputs and outputs of functions are valid. Zig's Debug build mode causes normally undefined behavior to result in a program crash, allowing programmers to patch up such scenarios.

Ease of Use

Each of the three languages make efforts to create code that is more readable and easier to use than C. Each language also has particular operations that are made especially convenient. We will walk through the ease of use features for each of the three languages.

First, D has multiple features that aim to make it easy to use. It has the same general "look and feel" of C, with the same compile-link-debug development model. Although D has pointers, it is designed to virtually eliminate their use for ordinary tasks. For example, features like array slicing and the `ref` keyword make it largely unnecessary to use raw pointers. D includes exception handling in the form of 'try-catch-finally' and built-in testing and verification tools. D has a variety of built-in data structures, like associative arrays (hash maps) and resizable arrays, that can be easily used. D's Vibe.d web framework allows for TCP/UDP connections, HTTP servers, and support for databases like MongoDB and Redis. Since our application involves connecting our cameras to a base station using a wireless network, this could be a useful framework, though there are security concerns since the framework is open source. Lastly, D documentation is highly readable due to its auto-generated documentation. D is the oldest of the three languages, and its official documentation website is detailed and easy to follow.

Next, Zig focuses on simplicity and has a great ease of use in that regard. Though it does not offer some of the additional features that D has, it will also likely be easier to pick up and learn. Zig has no hidden control flow that languages like D have with the `@property` attribute and operator overloading. One useful Zig feature are the `defer` and `errdefer` keywords, which defer certain actions until the end of the caller's scope. Other useful features are optional types, which prevent the need for null references, and the `comptime` keyword, which allows computations to run at compile time. These are powerful tools that simplify much of the C syntax. Zig also supports asynchronous functions in the form of the familiar keywords `async` and `await`. This can be highly useful for our system, where our base station will be performing I/O functions with each of the cameras.

Last, Odin has a number of specialized features as well. Arrays are very convenient to manage, with built-in slicing, dynamic arrays, and array computations. For example, we can call `a * b` for equally sized arrays a and b to perform a dot product. If our base station algorithm is performing SIMD-style calculations, this feature could be a natural fit. Some other Odin features include multiple results, named results, defer, type aliasing, and a built-in string type. Some elements of Odin are worrying, however. The presence of the `nil` type means that null reference errors are possible, unlike with the 2 alternative languages. In addition, the official Odin documentation is currently largely unfinished, with many blank sections. It may be difficult to learn to program in Odin for unfamiliar developers, especially since Odin is the youngest language out of the three options.

Performance

First, we will look at D's performance. It uses a highly optimized compiler and RAII, a modern software development technique to manage resource allocation and deallocation. D also has an inline assembler for when an application needs to dip into assembly language to get the job done. D includes the `synchronized` keyword for multithreading. One aspect of D that may harm its performance is the presence of a garbage collector. This garbage collector works by "pausing" a current thread to recursively scan for unreferenced objects and free them. These pauses are harmful to performance as they are nondeterministic, which can be unacceptable for a performance-critical application. If our customers require our SecureHEAT system to be highly responsive, this may be an issue. In addition, garbage collection in D is slower than other languages since it is unable to keep track of which pointers change between garbage collection cycles. This results in the entire heap needing to be scanned each time.

Next, we will explore Zig's performance. Zig runs faster than C and makes aggressive compile time optimizations in release mode to maximize runtime performance. In Zig, a single compilation unit with all included libraries is created before optimizations are run, which is in essence an automatic version of C's "Link Time Optimization". This results in a faster performance than calling dynamically linked functions. Zig's preprocessor also outperforms C's preprocessor by removing inconsistencies from calls to #include, #ifdef, etc. In Zig, several features like runtime-calculated array sizes and conditional compilation are allowed. There also exists an exposed SIMD vector type, which could be useful for parallel computations. Zig does not suffer from the inefficiencies of garbage collection like D does.

Last, although Odin is described as an "alternative to C with focus on performance, simplicity, and built for modern systems", it is difficult to find performance benchmarks. We will assume it does not have a significant advantage over C performance-wise.

Flexibility, Generality, and Reliability

Next, we will discuss flexibility, generality, and reliability for each language. First, D is designed to be a flexible language: it allows multiple paradigms, including function-and-data, object-oriented, template metaprogramming, and functional. D code is portable between compilers, OSes, and machines and custom garbage collection can be programmed. D programs can connect to any code that exposes a C ABI. D Code is made highly reliable due to the use of contract programming to set preconditions, postconditions, class invariants, and assert contracts. Unit tests can be made to run on program startup, verifying that class implementations haven't been broken on every build.

Zig has a great amount of flexibility with 4 different build modes that allow varying levels of compile time performance, runtime performance, and undefined behavior handling. For our application, the ReleaseSafe mode could provide a good mixture of safety checks with the tradeoff of a slightly slower runtime performance. Zig also allows us to interface directly with C by importing and exporting functions, which could be extremely helpful for interfacing with our camera drivers. In addition, programmers can choose whether to use libc or Zig's own standard library.

Finally, Odin has several features that lend itself to flexibility as well. It is convenient to interface with C and types like the built-in cstring were created with this purpose in mind.

Ethical Concerns

Next, we will discuss some of the ethical concerns in choosing a programming language for our application. The SecureHEAT system software is something that we must inherently take careful ethical consideration into. Since our customers will include clients in government, military, medicine, and other sensitive environments, we have an ethical responsibility to ensure that our system is secure. If sensitive information captured on our cameras (for instance, top secret government documents or patient health records) is obtained by an authorized party, we may be ethically responsible for allowing exploitable vulnerabilities in the system and the damages they cause. Therefore, the security of our selected programming language should play a large role in its selection. We will need to ensure that captured data is secure on several different levels. Captured video data should not be retained on the cameras, in case someone physically compromises the device. Information relayed throughout the wireless network to base stations should be encoded in case interceptors sabotage the network. Similarly, safety precautions must be taken if user temperature data is stored in some sort of database.

Another ethical concern is whether informed consent is given to occupants that inhabit SecureHEAT buildings. Those that occupy SecureHEAT buildings and have their thermal data filmed, recorded, and stored must be aware and understand how the system operates. Some may be uncomfortable with being filmed and having their information automatically stored for a purpose that previously did not require such obtrusive technologies. In order to help combat these sentiments, user data should be kept as private as possible and users should be given a thorough and transparent education on the technology. To aid with this, camera footage should be deleted as soon as it is no longer needed.

A final ethical concern is with the physical safety of users who engage with SecureHEAT. If, for instance, the system malfunctioned and raised the temperature to 100 degrees Fahrenheit, occupants could overheat and suffer health issues. Physical safety precautions must be kept in mind in order to prevent such dangerous conditions from occurring in the SecureHEAT system.

Conclusion

After weighing our three languages on various criteria and performing an ethical analysis on our system, we believe that Zig would make the best choice for the SecureHEAT system. D has the longest support and many useful features, but its complexity may prove difficult to use and the language may be too high-level for our use case. Odin suffers from having poor documentation and a niche that is not clearly defined, so it is difficult to evaluate for our use. Zig offers a mix of security, performance, and low-level programming that is ideal for interfacing with our camera hardware. Its different build modes offer programmers a great amount of flexibility in development while still resulting in highly optimized release builds. Its lack of a garbage collector and Debug mode ensure program security, an especially important feature considering the ethical implications of the product. Zig has a variety of features, including asynchronous functions and a ReleaseSafe mode that are great fits for SecureHEAT. Though it does not have a pre-existing wireless networking framework like D's Vibe.d, we may be unable to use such frameworks anyway due to the security risks involved in utilizing open source frameworks. Ultimately, Zig is a simple, highly performant, and secure language that makes it the best fit for SecureHEAT.

References

Cro, Loris. “What is Zig's Comptime?” 05 August 2019. https://kristoff.it/blog/what-is-zig-comptime/

Da Li, Carol C. Menassa, Vineet R. Kamat, Eunshin Byon. HEAT - Human Embodied Autonomous Thermostat. Building and Environment, 2020.

D High Level Overview. https://dlang.org/overview.html

D Language Specification. https://dlang.org/spec/spec.html

D Language Tour. https://tour.dlang.org/

Kelley, Andrew. Introduction to the Zig Programming Language. 8 February 2016. https://andrewkelley.me/post/intro-to-zig.html

Odin Language Overview. https://odin-lang.org/docs/overview/

Odin Language Specification. https://odin-lang.org/ref/spec/

University of Michigan. Turning faces into thermostats: Autonomous HVAC system could provide more comfort with less energy. ScienceDaily. ScienceDaily, 16 June 2020. www.sciencedaily.com/releases/2020/06/200616083353.htm

Zig Feature Highlights. https://ziglang.org/