self@myrrlyn.net https://myrrlyn.net Madison, WI +1 (517) 499-8820

Alexander Payne

Introduction

I am a software engineer specializing in low-level systems and freestanding code. I have over six years' experience developing embedded software for research satellites and experimental networking appliances, system software for high-traffic web servers, and foundation libraries in C, C++, and Rust. I have worked with electrical engineers to create novel hardware and with fleet operators to deploy high-performance software worldwide.

I excel at designing and implementing safety-critical code and proving conformance to external requirements, including both customer needs and formal engineering standards. I am a strong technical writer and produce my own documentation, for both technical and non-technical audiences. I am also an active contributor of open-source software, primarily in Rust. I maintain several widely-used utility libraries and lead the Ferrilab project.

Summary

These are explained with more detail on the following pages.

1. Sub-Zero/Wolf/Cove

I am a senior software engineer working on appliance control software.

2. Fastly

I was a senior software engineer responsible for creating traffic-management code in Rust and integrating it into a C codebase. My work was an ingress filter running worldwide on Fastly's fleet, and enabled significant reduction in malicious traffic without imposing a performance cost to any other users.

3. Amazon Web Services

I led the design and implementation of a Rust real-time unikernel running on a small ARM processor. I worked closely with electrical engineers to develop a novel networking appliance subject to strict performance, security, and tamper-resistance requirements.

4. Space Dynamics Laboratory

I wrote kernel modules and flight application software for small satellites, and earthside terminal software to control them. I also served as control staff for vehicle operations.

5. Ferrilab

I maintain an open-source Rust project which significantly reshapes the standard programming model. The `bitvec` crate is widely used throughout the public Rust ecosystem.

6. Formal Education

I studied Computer Engineering at Trine University. I learned digital system design in Verilog and embedded programming in C. I built and programmed a small autonomous vehicle for my thesis project.

self@myrrlyn.net https://myrrlyn.net Madison, WI +1 (517) 499-8820

Contents

self@myrrlyn.net https://myrrlyn.net Madison, WI +1 (517) 499-8820

1. Professional Experience

1.1. Sub-Zero/Wolf/Cove (2023–present) — Senior Software Engineer

I write realtime appliance control and interface software for kitchen appliances, primarily in C++ running on a suite of microprocessors.

1.2. Fastly (2021–2023) — Senior Software Engineer

I was the Rust subject-matter expert on a team maintaining the H2O web server powering Fastly's traffic ingress system. We worked on traffic management and prioritization, writing Rust modules and linking them into H2O's C codebase through an FFI bridge. Our work enabled Fastly to overcome DDOS attempts during the holiday season without a performance impact to our customers.

I was let go during the market contraction of the 2022-23 winter.

1.3. Amazon Web Services (2020–2021) — Software Development Engr

I was recruited to be the Rust subject-matter expert and software subteam lead for an experimental layer-2 networking appliance. I designed the software system architecture for a unikernel program running on an ARM Cortex-R processor, and implemented device drivers and the early application framework.

I am required to not disclose anything more about this project. It was cancelled after a year, and several of us chose to leave AWS when we weren't able to find suitable internal positions.

1.4. Space Dynamics Laboratory (2016–2020) — Satellite Software Engr

I wrote firmware and control software for space vehicles, wrote ground-station control software and performed on-orbit vehicle operations, and designed novel laboratory systems. I worked on both public and military projects, and the peculiar nature of satellite operations has vastly over-prepared me for terrestrial industry.

I resigned from SDL after my grandmother's death and mother's illness early in the COVID-19 pandemic so that I could spend time supporting my mother in her recovery.

1.4.1. GRYPHON (2020) – AFRL

I worked on the design and initial standup of an experimental laboratory and simulation environment. This was a classified project, and I cannot provide details about my tasking.

1.4.2. VPM (2019) — AFRL

I wrote vehicle control software in C++11, using SDL's RADIANT framework.

1.4.3. EAGLE (2018) — AFRL

I wrote ground-station control software in Python 2 and supplied continuous updates for over a year until the project was relocated away from Kirtland AFB. This was a classified mission and I was not permitted to be present in the SCIF during vehicle operations. As such, deployment required carrying patches back and forth between my office and the SCIF, and manually updating the software both in production and on my development machine.

1.4.4. DHFR (2017) — DARPA

I wrote ground-station control software in Ruby, using the COSMOS (now OpenC3) framework and assisted with post-assembly vehicle testing. I then performed vehicle launch-and-early-operations for the mission until the vehicle was declared dead on orbit.

The orbital characteristics of the DHFR mission gave it a 9-hour window of periodic visibility over our ground-station network, followed by 14 hours where it was not visible. The resulting 23-hour "day" meant that my shift began one hour earlier every day, rotating backwards around the clock. I was the sole continuous staffer for the four months between launch and end of mission.

1.4.5. BioSentinel (2017) — NASA

I wrote a kernel module in C99 for VxWorks 6 on a SPARC v8 chip. This module multiplexed a number of discrete hardware sensors aboard an FPGA over a single SpaceWire connection, allowing userland software running on the CPU to access each of the sensors through a named device file.

Due to the design of the SPARC architecture and NASA's restrictions on dynamic memory allocation, this required careful memory management and an implementation with as few interior function calls as possible.

2. Public Works (2018–present)

I am an active contributor to the Rust language's open-source collection. I also write some Elixir and TypeScript web applications.

2.1. bitvec

`bitvec` implements bit-precision addressing as an ordinary library. It provides idiomatic collections and behavior, including arrays, dynamic vectors, and borrowed slices. It allows client code to specify both the integer type used for backing storage and the order of bits within those integers, implements the entire standard-library sequence API, and is entirely thread-safe.

This project pushes the boundaries of what the Rust language is able to express, and is a case study in ongoing development of the Rust abstract machine's pointer model.

2.2. funty

`funty` provides traits that abstract over the Rust primitive types, allowing client code to become generic over the primitives while still retaining access to their full API. This allows, for instance, client code to become generic over the width of a numeric type but making use of properties such as signedness.

2.3. radium

`radium` unifies Rust's shared-mutability markers. `Cell` is not thread-safe but requires no special hardware support; the `atomic` module contains types that are thread-safe but are not guaranteed to exist on every `radium` allows code to defer which of these families is used to provide shared mutability. It provides best-effort type aliases that resolve to atomics when present and cells when not, allowing code to become portable across different targets without incurring compiler errors.

2.4. tap

This library provides convenience methods that allow common operations (inspection, mutation, or conversion) to be placed in suffix-call position. This is analagous to Elixir's pipe operator (|>), or D's implementation of Universal Method-Call Syntax.

2.5. calm_io

This provides alternatives to Rust's standard-stream write macros that do not panic on error, and a decorator for `fn main` which detects when `main` returns with `io::ErrorKind::BrokenPipe` and converts it into a graceful exit.

Without this crate, any Rust program which uses `println!` can be induced to panic by running it as `prog | head -n0`: this argument causes `head` to quit immediately, `prog`'s stdout stream closes, and `println!` unwraps the error returned when writing to a closed pipe.

The Rust project is currently working on integrating this behavior into the standard library.

3. Skills and Abilities

- I am an expert Rust programmer, fluent in C++11 and C99, and am capable with Ruby.
- I am specialized in asymmetrically-distributed systems and CLI tools, and familiar with web applications. I have not written desktop graphical software professionally.
- I produce my own technical writing, including both internal API documentation, user manuals, and engineering reports.
- I primarily use Git and Linux. I have used Docker for both development environments and application deployments. I am familiar with Mercurial and Windows PowerShell, but have not used them extensively.
- My public work is on GitHub; I have also used GitLab and the Atlassian suite professionally.
- I can rapidly learn unfamiliar systems and technologies. I have enough of an electrical engineering background to follow along with work in that area, but I am no longer able to work with hardware beyond writing Verilog modules.

4. Formal Education

4.1. B.Sc., Computer Engineering, Trine University (2016)

My thesis project was the construction, programming, and operation of an autonomous freight vehicle. My responsibilities on the team were:

- drivetrain component selection and assembly
- control system component selection and assembly
- freestanding control software design and implementation
- device driver implementation for:
  - GPS receiver (positioning)
  - magnetometer (orientation)
  - ultrasonic sensors (environmental awareness)
  - motor controllers (movement)
  - axle Hall-effect sensors (closed-loop PID control)

I also studied digital component design, including the construction of a MIPS CPU in Verilog which was required to execute real programs when programmed to an FPGA.

5. Additional Qualifications

5.1. Federal Clearance

I have held a TS/SCI clearance since 2018. I was last read out in 2021 August, and so my investigation is expired as of 2023 August. I maintain a lifestyle that is conducive to clearance investigations should the need arise.

5.2. Aquatics

I am a PADI rescue diver and Scouts BSA lifeguard instructor. I believe strongly in the importance of imposing safety onto a hazardous environment, and bring this focus to all aspects of my work.