CS340R: Rusty Systems

Spring 2024
Instructor: Philip Levis
CAs: Rae Wong and Deepti Raghavan

WIKIPEDIA
The Free Encyclopedia

Create account Log in

Rust for Linux

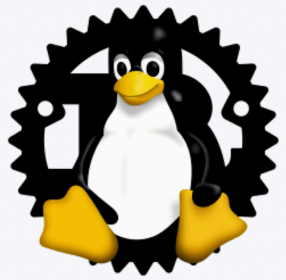
Add languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

Rust for Linux is a series of patches to the Linux kernel that adds Rust as a second programming language to C for writing kernel components.

Rust for Linux



Developer(s)	Community contributors Miguel Ojeda
Repository	github.com/Rust-for-Linux/linux
Written in	Rust
Available in	English
License	GPL-2.0-only with Linux-syscall-note.
Website	https://rust-for-linux.com/

History [edit]

The Linux kernel has been primarily written in C and assembly language since its first release in 1991. Around 1997, the addition of C++ was considered and experimented upon for two weeks before being scrapped.[1] Rust was created in 2006 and combines the performance of low-level programming languages (such as C) with a focus on memory safety and a user-friendly tool set and syntax.[2]

The Rust for Linux project was announced in 2020 in the Linux kernel mailing list with goals of leveraging Rust's memory safety to reduce bugs when writing kernel drivers.[3] At the Open Source Summit 2022, Linus Torvalds stated that the incorporation of the project's work could begin as soon as the Linux 5.20 release, later named as Linux 6.0.[4] The first release candidate for Linux 6.0 was created on 14 August 2022, without Rust support. In the release notes for Linux 6.0-rc1, Torvalds expressed his intention for adding Rust support, "I actually was hoping that we'd get some of the first rust infrastructure, and the multi-gen LRU VM, but neither of them happened this time around."[5][6] On 19 September 2022, an article from ZDNet revealed an email from Linus Torvalds stating that "Unless something odd happens, it [Rust] will make it into 6.1".[7]

In October 2022, a pull request for accepting the implementation for Rust for Linux was approved by Torvalds.[8] As of Linux 6.1, support was intentionally left minimal in order to allow developers to test the feature.[9]

References [edit]

1. ^ Claburn, Thomas (2022-06-23). "Linus Torvalds says Rust is coming to the Linux kernel". *The Register*. Archived from the original on 2022-07-28. Retrieved

Microsoft is busy rewriting core Windows code in memory-safe Rust

Now that's a C change we can back

Thomas Claburn

Thu 27 Apr 2023 // 20:45 UTC

Microsoft is rewriting core Windows libraries in the Rust programming language, and the more memory-safe code is already reaching developers.

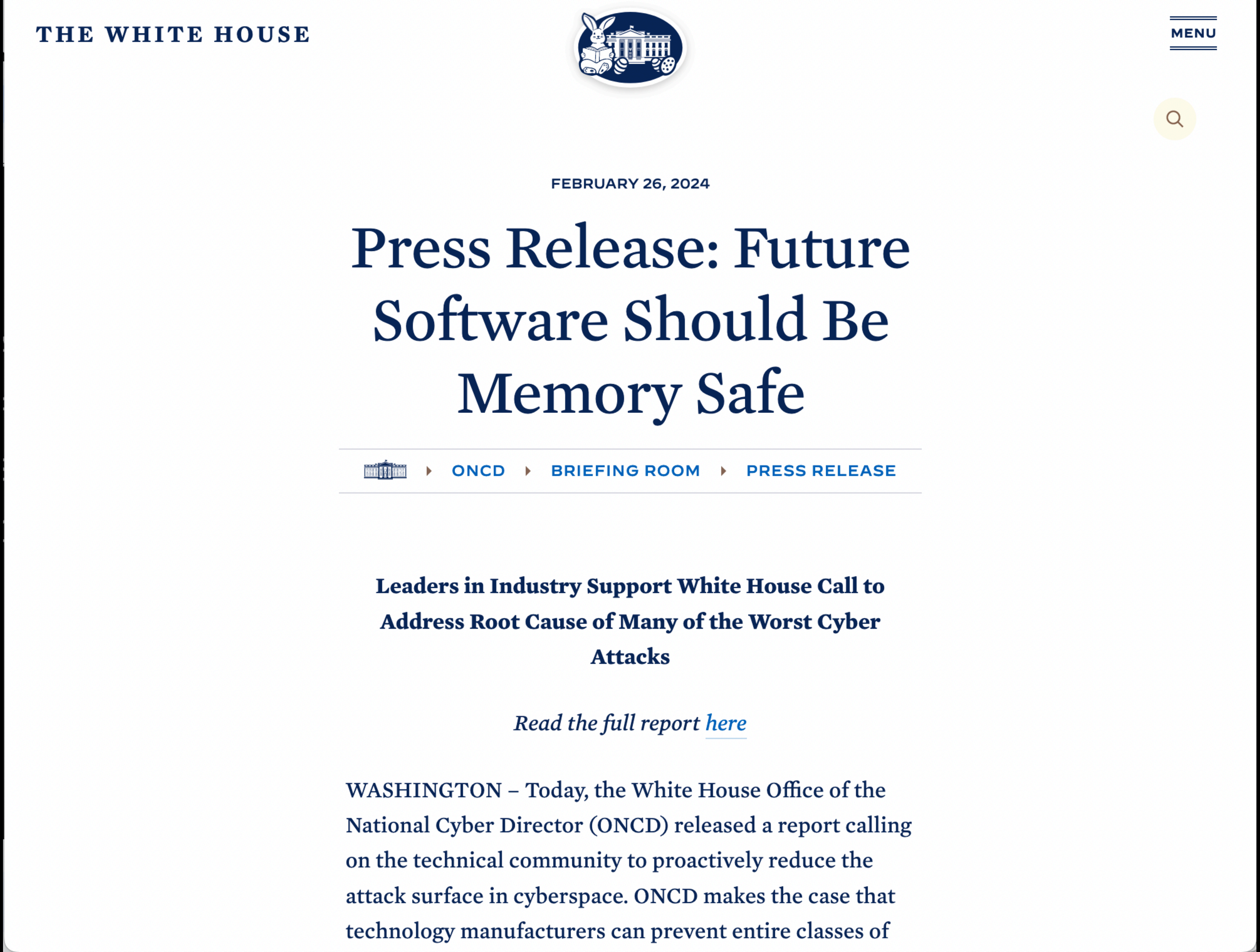
David "dwizzle" Weston, director of OS security for Windows, announced the arrival of Rust in the operating system's kernel at BlueHat IL 2023 in Tel Aviv, Israel, last month.

"You will actually have Windows booting with Rust in the kernel in probably the next several weeks or months, which is really cool," he said. "The basic goal here was to convert some of these internal C++ data types into their Rust equivalents."

Microsoft showed interest in Rust several years ago as a way to catch and squash memory safety bugs before the code lands in the hands of users; these kinds of bugs were at the heart of about 70 percent of the CVE-listed security vulnerabilities patched by the Windows maker in its own products since 2006.

The Rust toolchain strives to prevent code from being built and shipped that is exploitable, which in an ideal world reduces opportunities for miscreants to attack weaknesses in software. Simply put, Rust is focused on memory safety and similar protections, which cuts down on the number of bad bugs in the resulting code.

Rivals like Google have already publicly declared their affinity for Rust.



FEBRUARY 26, 2024

Press Release: Future Software Should Be Memory Safe

Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

Read the full report here

WASHINGTON – Today, the White House Office of the National Cyber Director (ONCD) released a report calling on the technical community to proactively reduce the attack surface in cyberspace. ONCD makes the case that technology manufacturers can prevent entire classes of

Yes, but...

Metanarrative

What follows is a kind of "metanarrative" of using async Rust that summarizes the challenges that are present today. At each point, we link to the various stories; you can read the full set in the table of contents on the left. We would like to extend this to also cover some of its glories, since reading the current stories is a litany of difficulties, but obviouly we see great promise in async Rust. Note that many stories here appear more than once.

Rust strives to be a language that brings together performance, productivity, and correctness. Rust programs are designed to surface bugs early and to make common patterns both ergonomic and efficient, leading to a sense that "if it compiles, it generally works, and works efficiently". Async Rust aims to extend that same feeling to an async setting, in which a single process interweaves numerous tasks that execute concurrently. Sometimes this works beautifully. However, other times, the reality falls short of that goal.

▸ Making hard choices from a complex ecosystem from the start
▸ Once your basic setup is done, the best design patterns are subtle and not always known.
▸ Even once you've chosen a pattern, gettings things to compile can be a challenge.
▸ Once you get it to compile, things don't "just work" at runtime, or they may be unexpectedly slow.
▸ When you have those problems, you can't readily debug them or get visibility into what is going on.
▸ Rust has always aimed to interoperate well with other languages and to fit itself into every niche, but that's harder with async.

If someone tells you Rust will solve all your problems, ask them if they've ever programmed in Rust.

Sapir-Whorf Hypothesis

- Language shapes thought
- This is true for natural languages (Spanish vs. Mandarin vs. English)
- It is true for computer languages too
  - String processing code in C vs. Python
  - Concurrency in C vs. Python

C/C++

- For 40 years, C/C++ defined computer systems
- Defined what was easy and what was hard: defined research challenges
  - Address space layout randomization
  - Bug finding
- But now we have new languages (Rust, Go): how do problems change?

Mismatch in Research

- Just write it in Rust -- it's safe and fast!
- But very few faculty know (or care to know) Rust...
- Using Rust with lots of unsafe isn't really using Rust
  - Sacrifice its guarantees to just "make things work"

This Class

What are the most important open research challenges for software systems written in Rust?

300-level class

- This isn't a "I love Rust" class
- This isn't a "I want to learn Rust" class
- This isn't a "I want to learn Rust better" class
- This *is* a class about exploring, understanding, and defining the research problems that Rust introduces

Research Problems

- You should have a graduate-level understanding of systems
- You should have experience with systems research
  - CS240, CS240LX, etc.
  - CS244
  - CS244B
- You should have a good understanding of systems abstractions, techniques, and problems
- You should understand what's research and what's engineering

Class Structure

- First 3 weeks
  - Crichton and Krishnamurti's modified Rust Book: learn/relearn Rust, do all the exercises
  - Reading short (HotOS/APSys) papers discussing Rust possibilities and challenges
- Next 7 weeks
  - Working in groups of 3-4 on a project: re-implement an existing (C/C++) research system in Rust and find out what's difficult
  - Read full-length research papers on Rust or that use Rust
- Class time is spent discussing papers
- CAs are here to help you with Rust challenges
- Grade is 80% project and 20% class participation