Use of Rust Language for the Implementation of more secure, reliable, high performance Operating System

Content

Linux
Apple
Microsoft Windows
IBM
ubuntu
UNIX
android
MS DOS
MS DOS
Mac OS
BlackBerry
OS/2 WARP

1. What is an Operating System?

- Operating System is the program that manages the computer hardware.
  It also provides a basis for application programs and act as the intermediary between the computer user and the computer hardware.

- The operating system defines the liveness of the machine: without it, no program can run

- The operating system software that runs with the highest level of architectural privilege is the operating system kernel

- …but the kernel is not the entire operating system!

2. About Rust

- Rust is a systems software programming language designed around safety, parallelism, and speed

- Rust was originally designed by Graydon Hoare at Mozilla Research on July 7, 2010.

- Rust has a novel system of ownership, whereby it can statically determine when a memory object is no longer in use

- This allows for the power of a garbage-collected language, but with the performance of manual memory management

- This is important because — *unlike C* — Rust is highly composable, allowing for more sophisticated (and higher performing!) primitives

3. Performance

- Rust is blazingly fast and memory-efficient:
  - ✓ with no runtime
  - ✓ garbage collector

- It can power,
  - performance-critical services
  - run on embedded devices
  - easily integrate with other languages

Runtime;

- Runtime system is the first function runs when we run a program.
- A very few of runtime owned to the Rust.
- Part of the standard library can be considered as runtime.
- Code can be compiled without standard library.

Garbage Collector;

GARBAGE COLLECTION (GC), ALSO KNOWN AS AUTOMATIC MEMORY MANAGEMENT, IS THE AUTOMATIC RECYCLING OF DYNAMICALLY ALLOCATED MEMORY.

THE SINGLE-OWNER-RESTRICTION ALLOWS RUST TO ENFORCE MEMORY SAFETY WITHOUT A GARBAGE COLLECTOR.

RUST'S AUTOMATIC MEMORY MANAGEMENT IS FULLY STATIC.

OWNERSHIP AND BURROWING SYSTEMS HELPS TO AVOID IT.

4. Reliability

- Rust's
  - ✓ rich type system
  - ✓ memory model

guarantee memory-safety and thread-safety

- This enabling you to eliminate many classes of bugs at compile-time.

Type System;

- A statically typed language.
- A value in Rust can be defined by the type and representation in memory.
- Any number of traits can implement using a single type
- M2M relationship between traits and implementing types allows programmers to express interface inheritance
- “trait-object” in Rust is the way to achieve dynamic dispatch.
- Compiler will generate a set of “Vtables”, to find the correct methods at runtime

Memory Model;

- Similar to the C's memory model and a stricter version of it.
- Compiler statically checked the lifetimes to prevents all use-after-free bugs.
- Rust has “raw pointers”, which behave more like C counterparts,…but raw pointers are not safe to use.
- Therefore, “unsafe” keyword to separate this unsafe code from the rest, allowing the programmers to write data structures which use unsafe code internally while providing a safe interface to the user.
- The “single-owner-restriction” can enforce memory safety.

5. Productivity

- Rust has
  - ✓ great documentation
  - ✓ a friendly compiler with useful error messages (Error Handling)
  - ✓ macros for printing text and initializing vectors

- Smart multi-editor support with autocompletion and type inspections, an auto-formatter, and more.

Error Handling;

- Mechanisms used for dealing with errors in Rust;
  - ✓ Return values
  - ✓ Stack unwinding

- There is also an exception handling system for more unexpected failures in Rust…but it is not the default mechanism for handling errors in Rust.

- “panic handler” aborts the current operation in a memory-safe way when the ‘panic!’ macro or assertion failure is raised.

- The stack unwinding provides memory safety by correctly reclaiming only “out-of-scope” objects.

Macros;

- Macros can simply define as a list of patterns and expansions.
- When a macro is used, these patterns are matched against the arguments given, and the corresponding expansions used as substitutions for the invocation.
- Rust code can be transformed with use of macros or compiler plugins, at compilation time.
- Rust macros are applied on the Abstract Syntax Tree (AST), rather than on the source code.
- Rust uses macros for printing text and initializing vectors.

6. Conclusion

- Rust has a number of other features that make it highly compelling for systems software implementation:

  - ✓ Algebraic types allow robust, concise error handling
  - ✓ Hygienic macros allow for safe syntax extensions
  - ✓ Foreign function interface allows for full-duplex integration with C without sacrificing performance
  - ✓ “unsafe” keyword allows for some safety guarantees to be surgically overruled (though with obvious peril)

Thank you