Rust para Linux Embarcados

Jonathas A. O. Conceição

Dezembro - 2019 - Porto Alegre/RS

Sumário

Introdução

Limitações dos sistemas embarcados

- Espaço em armazenamento;
- Memória principal;
- Previsibilidade do uso de memória;
- Arquitetura de Hardware;

Porque usar Rust

- Portabilidade;
- Controle do uso de Memória;
- Compatibilidade externa;
- Código seguro;
- Comunidade;

Imagens customizadas

Limitações

- Bibliotecas de sistema;
- Acesso direto à dispositivos de Hardware;
- Compilação cruzada;
- Previsibilidade do uso de memória;
- Uptime;

Imagens personalizadas

- Yocto Project
- BuildRoot;
- Android;

Porque usar Rust

Portabilidade

- Compilação condicional;
- Opções nativas;
- Prelude opcional;
- Compilação com o LLVM;
- Dedicação da comunidade;

Exemplos

Compilação condicional

```
1  #[cfg(target_os = "linux")]
2  fn are_you_on_linux() {
3      println!("You are running linux!");
4  }
5
6  #[cfg(not(target_os = "linux"))]
7  fn are_you_on_linux() {
8      println!("You are *not* running linux!");
9  }
```

Porque usar Rust

Controle do uso de Memória

- Código seguro;
- Alocação dinâmica é opcional;
- Controle de efeito colateral;
- Tratamento de erro explícito;

Exemplos

Tratamento de erro

```
1  let f = File::open("hello.txt");
2
3  let f = match f {
4      Ok(file) => file,
5      Err(error) => {
6          panic!("Problem opening the file: {:?}", error)
7      },
8  };
```

Porque usar Rust

Compatibilidade externa

- Bindings FFI;
- Palavra reservada `unsafe` para código inseguro;
- Facilitadores como o rust-bindgen e cbindgen;

Exemplos

Usar Rust em C

```
1  #[no_mangle]
2  pub extern "C" fn double_input(input: i32) -> i32 {
3      input * 2
4  }
```

Usar C em Rust

```
1  extern "C" {
2      fn double_input(input: libc::c_int) -> libc::c_int;
3  }
```

Porque usar Rust

Comunidade

- Alto engajamento nas filosofias da linguagem;
- Desenvolvimento comunitário e aberto;
- Amigabilidade;

Gerenciador de instalação

rustup

- Instalação de *toolchains*;
- Atualização de ferramentas;
- Instalação de componentes;

Gerenciador de pacotes

Cargo

- Dependências;
- *Builds*;
- Testes;
- *Banchmarks*;

Exemplos

Cargo.toml

```
1  [package]
2  name = "exemplo"
3  edition = "2018"
4  build = "src/build.rs"
5
6  [lib]
7  name = "libexterna"
8  crate-type = ["lib", "staticlib", "cdylib"]
9
10 [dependencies]
11 crc-any = "2"
12 flate2 = { version = "1", default-features = false, features =
       ["zlib"] }
13
14 [build-dependencies]
15 git-version = "0.3"
```

Gerenciador de código

rustfmt

- Formatação de código;
- Parametrizável;

rust-clippy

- Dicas de bons padrões de código;

Ferramenta para compilação cruzada

cross

- Gerenciamento de *containers* para compilação cruzada;

Conclusão

Pontos altos

- Ótimas ferramentas;
- Ótimo desempenho;
- Código seguro;

Pontos à serem melhorados

- Alta complexidade de código;
- Compilação lenta;
- Microdependências;
- Maturidade;
  - libusb;
  - libarchive;

Dúvidas? Comentários?

Links relevantes

- **Instalação:** `rustup.rs`
- **Pacotes da comunidade:** `crates.io`
- **Livro:** `doc.rust-lang.org/book`
- **Compilação cruzada:** `github.com/rust-embedded/cross`
- **Cheat Sheet:** `cheats.rs`

Mais informações:

E-mail: `jonathas.conceicao@ossystems.com.br`