OS中期报告——RustOS for x86_64 SMP

2018.04.12

王润基 朱书聪

选题概述

- 将ucore用Rust重写，使之更加安全和模块化。
- 参考sv6进行SMP优化，使用Commuter进行测试。

Why Rust?

- 系统级编程语言
- 安全内存管理模型，保证线程安全
- 支持不安全的底层操作，能和C语言方便交互
- 零开销抽象，拥有现代语言特性

——适合编写OS！适合处理并发问题！

已经完成的调研工作

- Rust for RISCV
- RustOS
- 待移植/参考OS
- Rust与编写OS相关的平台特性

Rust for RISCV

- RISCV-Rust-Toolchain：
  - Docker：tarcieri/riscv-rust-toolchain
  - 与之配合的开发环境：riscv-crates
  - 目前无法编译，还在开发中

鉴于这套工具链刚刚诞生，非常不稳定，坑太多，因此我们不再考虑移植到RISCV。

RustOS

- 《Writing an OS in Rust》& blog_os：
  - 从零开始写RustOS的教程，平台x86_64
  - 目前实现了Boot，页式内存管理，中断管理
  - 已经读完了全部文章，能够在本机和Docker中编译运行
- Redox：
  - 完成度最高的RustOS，微内核架构，平台x86_64
  - 有GUI，有自己的FS，有用户程序开发环境
  - 正在阅读kernel代码，已经在Docker中完成编译

待移植/参考OS

- xv6：ucore的起源项目，非常精简，但支持SMP。用来理解OS的整体运行机制。
- xv6 x86_64：对比与xv6的不同，参考其x86_64的实现。
- ucore_os_lab：待移植的主体。
- ucore_plus：参考其x86_64的实现。
- sv6：参考其多核优化的实现。

Rust与编写OS相关的平台特性

库：

- 标准库std ×，核心库core √
- core提供了最基础的语言特性支持，字符串格式化输出等基础函数，只需指定全局内存分配器，即可享用Vec/Box等大量内置数据结构

内联汇编：

- 格式类似C语言
- x86(_64)已经有封装好的库，无需手动编写

Rust-C互操作：

- C语言绑定：
  - bindgen工具：自动生成C模块的Rust接口
  - 直接导入符号
- C对Rust的绑定：
  - 通过简单的`extern "C"`+`no_mangle`即可完成
- C直接生成Rust：Corrode
  - 在macOS上尝试构建失败
- Port C to Rust 经验谈：
  - https://github.com/carols10cents/rust-out-your-c-talk

- 测试：
  - `cargo test`单元测试，`cargo bench`性能测试
  - 由于OS运行在qemu环境中，因此无法使用这个测试框架
  - 但已经在Travis上配置了集成测试（类似ucore）
  - 可以把不依赖OS环境的模块提取成crate，就可以测试了
- 坑：
  - 除了经常过不了编译以外，还没遇到大坑。
  - Rust大法好。

已经完成的编码工作

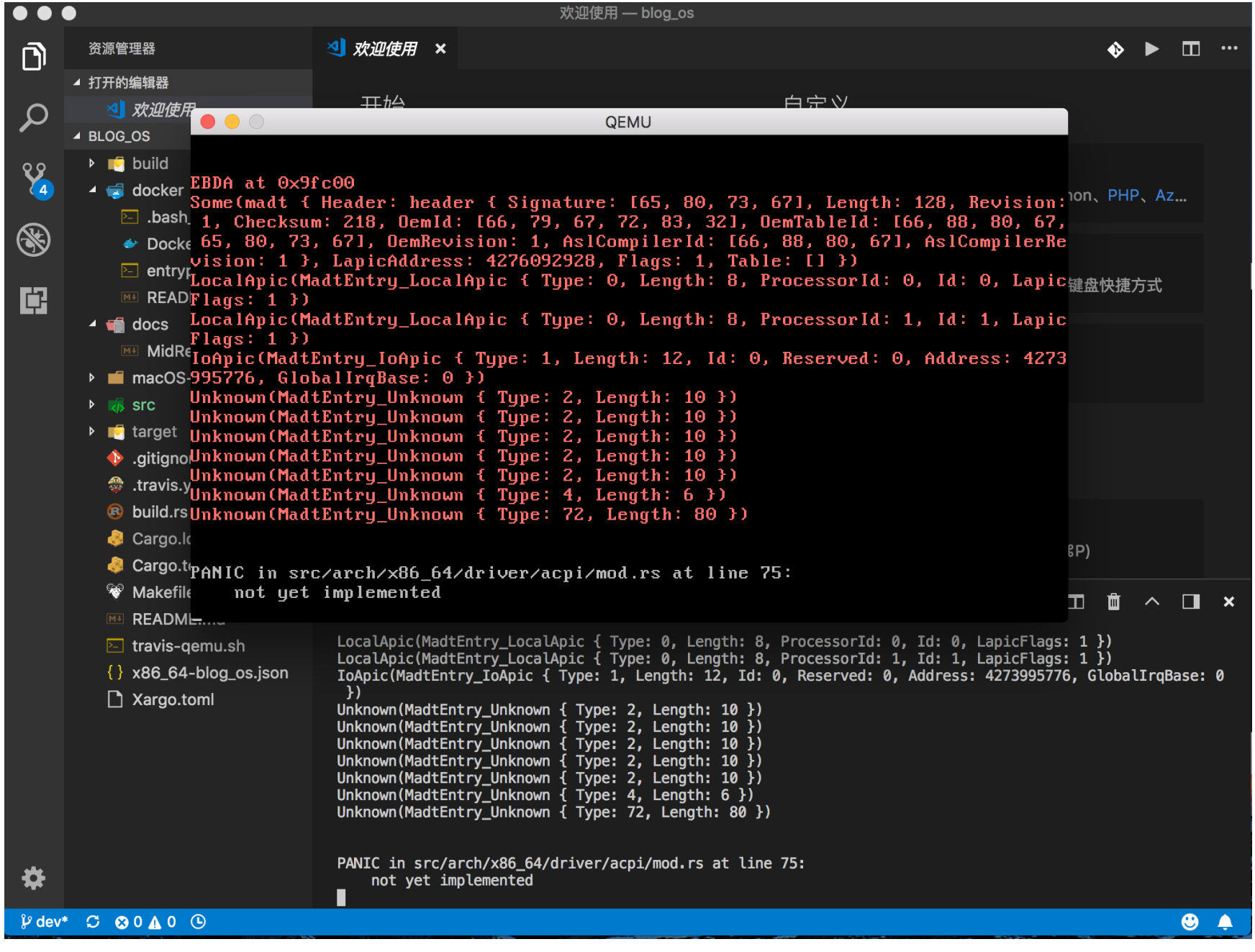
- 准备开发环境

  根据对RustOS的调研情况，我们计划直接在blog_os的基础上进行开发。目前已经配置好了本地环境，Docker，Travis。

- 熟悉Rust语言和生态特性

  实现了C语言绑定，在线集成测试。

  已经完成了一些OS底层驱动的移植（VGA，Serial，ACPI）。


欢迎使用 — blog_os
资源管理器
欢迎使用
打开的编辑器
欢迎使用
BLOG_OS
build
docker
.bash
Docke
entryp
READ
docs
MidRe
macOS
src
target
.gitigno
.travis.y
build.rs
Cargo.lo
Cargo.to
Makefile
README
travis-qemu.sh
x86_64-blog_os.json
Xargo.toml
QEMU
EBDA at 0x9fc00
Some(madt { Header: header { Signature: [65, 80, 73, 67], Length: 128, Revision: 1, Checksum: 218, OemId: [66, 79, 67, 72, 83, 32], OemTableId: [66, 88, 80, 67, 65, 80, 73, 67], OemRevision: 1, AslCompilerId: [66, 88, 80, 67], AslCompilerRevision: 1 }, LapicAddress: 4276092928, Flags: 1, Table: [] })
LocalApic(MadtEntry_LocalApic { Type: 0, Length: 8, ProcessorId: 0, Id: 0, LapicFlags: 1 })
LocalApic(MadtEntry_LocalApic { Type: 0, Length: 8, ProcessorId: 1, Id: 1, LapicFlags: 1 })
IoApic(MadtEntry_IoApic { Type: 1, Length: 12, Id: 0, Reserved: 0, Address: 4273995776, GlobalIrqBase: 0 })
Unknown(MadtEntry_Unknown { Type: 2, Length: 10 })
Unknown(MadtEntry_Unknown { Type: 2, Length: 10 })
Unknown(MadtEntry_Unknown { Type: 2, Length: 10 })
Unknown(MadtEntry_Unknown { Type: 2, Length: 10 })
Unknown(MadtEntry_Unknown { Type: 2, Length: 10 })
Unknown(MadtEntry_Unknown { Type: 4, Length: 6 })
Unknown(MadtEntry_Unknown { Type: 72, Length: 80 })
PANIC in src/arch/x86_64/driver/acpi/mod.rs at line 75:
not yet implemented
dev*
0 0

接下来需要做的工作

综述

blog_os只实现了页式内存管理和中断，提供了一个RustOS的基础框架。在此基础上，针对ucore的每个lab，还需补全以下内容：

- **lab1：**
    - 修改虚拟地址的映射，使之与ucore匹配。
    - 补全底层驱动（键盘、串口、APIC），重点是能产生时钟中断。
        - 这部分大都是直接操作内存，可以直接链接C代码，以尽快跑起来，之后再用Rust重写。

- lab2/3：内存管理
    - 物理内存分配器目前只分配不回收，需要完善
    - 页式内存管理没有实现替换机制，需要实现

  注：由于只是要完善，不影响后面的工作
- **lab4/5/6：进程管理**
    - 进程管理和调度需要重新设计、整体移植
- lab7：同步互斥
    - 基本的同步机制Rust标准库已经提供
- **lab8：文件系统**
    - 对底层依赖不大，可以独立出去完成（参考Redox的FS）

支持SMP主要依赖进程管理模块，文件系统部分委托合作小组完成。

分工

- 王润基：主要负责前期ucore移植
- 朱书聪：主要负责后期SMP优化

计划

- Week7：完成底层硬件的移植，能跑lab1
- Week8：完善物理/虚拟内存管理，能跑lab2/3
- Week9：移植内核线程、用户进程、调度，能跑lab4/5/6
- Week10：支持SMP，配置Commuter生成代码
- Week11-13：SMP优化