

RuCore with LKM Drivers

g15 杨国炜 乔一凡

课程设计目标

- 使用 Rust 重新实现 ucore，目标平台为 x86_64
- 为移植好的 ucore 适配相应的驱动，包括如下几部分：
 - 存储设备驱动，ATA/SATA
 - 显示设备驱动，VGA图形驱动
 - PS2 键盘/鼠标驱动
- 在 64 位 ucore 上实现可加载内核模块（LKM），并实现驱动的模块化

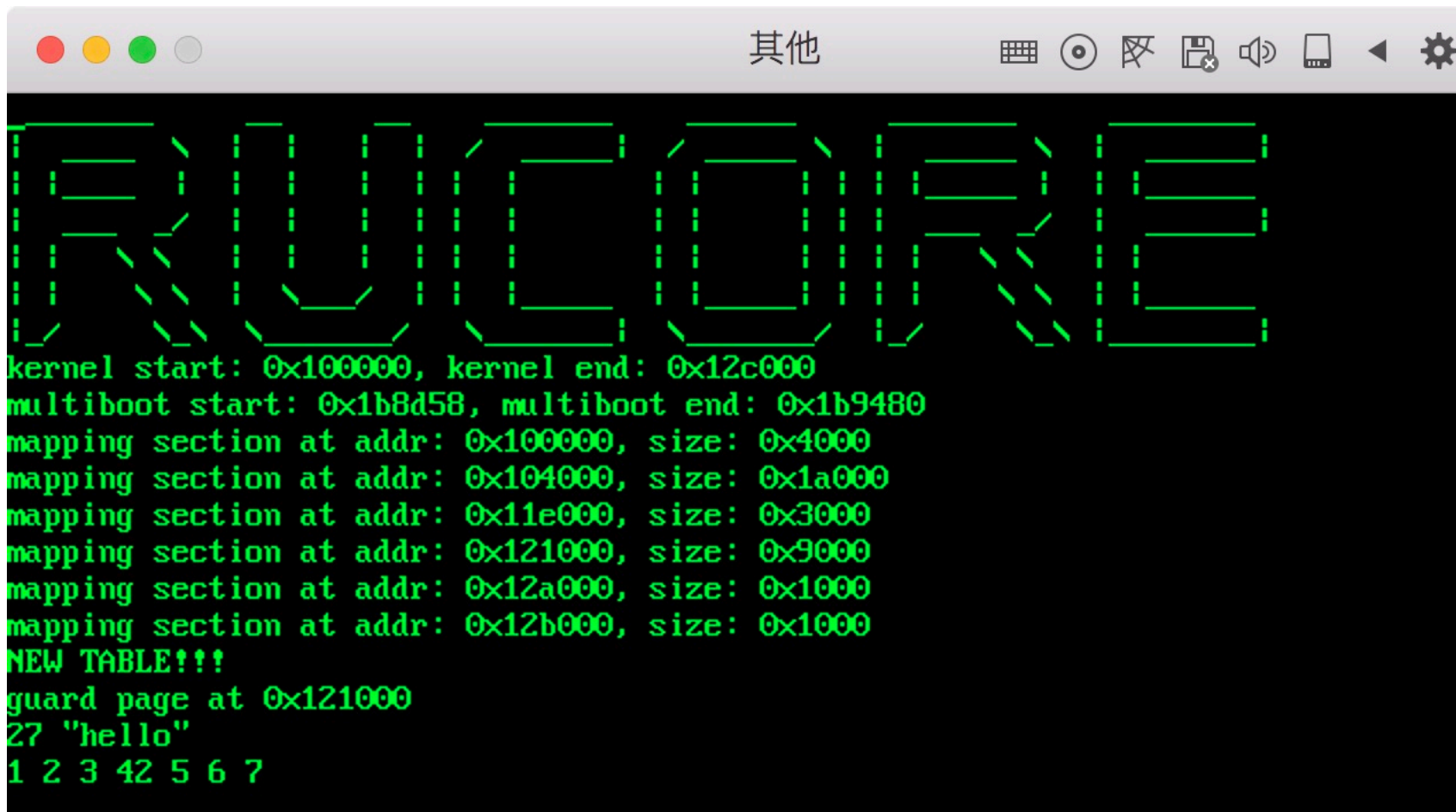
Why Rust ?

- 与 C 相比, Rust
 - 有 C 的优点: 静态类型, 编译语言, 没有 runtime,
 - 灵活: `unsafe block`, 支持与 C 的混合编程, 调用 C 函数, 易于嵌入汇编
 - 安全:
 - 内存安全: 禁止数据竞争, 解引用空/裸悬垂指针

目前工作进度

- 阅读博客 [Writing an OS in Rust](#)，并根据博客内容实现了简单的 `blog_os`，主要功能如下：
 - 系统的启动，进入保护模式以及 `x86_64 long mode` 的切换
 - 对显存的基本操作，能够通过 `VGA` 显示字符
 - 基本的内存管理，四级页表的建立，虚拟内存
 - 中断向量表的建立，实现简单的异常处理

效果展示



A screenshot of a terminal window with a macOS-style title bar. The title bar contains three colored window control buttons (red, yellow, green) on the left, the text '其他' (Other) in the center, and a series of system utility icons (keyboard, mouse, network, storage, volume, display, and settings) on the right. The terminal itself has a black background with green text. At the top, the word 'RUNNING' is displayed in large, hollow, green capital letters. Below this, several lines of text provide system information: 'kernel start: 0x100000, kernel end: 0x12c000', 'multiboot start: 0x1b8d58, multiboot end: 0x1b9480', and a list of memory mapping sections with their addresses and sizes. This is followed by 'NEW TABLE!!!', 'guard page at 0x121000', '27 "hello"', and a sequence of numbers '1 2 3 42 5 6 7'.

```
kernel start: 0x100000, kernel end: 0x12c000
multiboot start: 0x1b8d58, multiboot end: 0x1b9480
mapping section at addr: 0x100000, size: 0x4000
mapping section at addr: 0x104000, size: 0x1a000
mapping section at addr: 0x11e000, size: 0x3000
mapping section at addr: 0x121000, size: 0x9000
mapping section at addr: 0x12a000, size: 0x1000
mapping section at addr: 0x12b000, size: 0x1000
NEW TABLE!!!
guard page at 0x121000
27 "hello"
1 2 3 42 5 6 7
```

实现计划

- 根据之前的三个小目标逐步实现
 1. 使用 **Rust** 实现 **ucore**
 2. 移植相应驱动
 3. 支持 **LKM** 并模块化驱动

使用 **Rust** 重新实现 **ucore**

- Week 7-9
- 参考工作：
 - 已有的 `blog_os` 的基本框架
 - `ucore on x86_64` 工程
 - `Reenix, Redox, etc.`: 参考其一些 **Rust** 的底层实现方式

使用 Rust 重新实现 ucore

- 总体计划按照 ucore lab 的顺序，逐步重新实现 lab1-8 的功能，最终完成移植
- 小组间合作计划
 - 1.前期 ucore 的移植部分由几个小组合作完成，通过与 C 代码混合编译的方式完成 lab1 的基本功能
 - 2.之后同时进行进程创建和调度（lab4-6, by g11）和文件系统（lab8, by g15）
 - 3.合并代码，完善虚拟内存管理（lab3），将 lab1 中代码用 rust 实现，完成整体移植

使用 Rust 重新实现 ucore

- 组内开发计划
 - **Lab1**：需要实现一系列底层驱动，包括 PIC, PIT, UART 等等，已经有 ucore 的实现（乔一凡）
 - **Lab8** 文件系统：与平台相关性不大，仿照 ucore 的思路，将文件架构抽象成四层并分别实现（高两层：乔一凡；低两层：杨国炜）
 - **Lab3** 虚拟内存管理：已建立分页机制和简单的异常处理，需要实现页替换算法（杨国炜）

适配驱动

- Week 10
- 存储设备驱动（杨国炜）
 - ucore 已有 IDE 驱动，也就是 ATA 接口映射模式的驱动
 - 移植 ATA，同时实现类似的 SATA 接口的 ACHI 模式驱动
- PS2 键盘，鼠标驱动（乔一凡）
 - ["Tifflin" Experimental Kernel](#)：包含一些基本的控制器驱动，可以参考其与底层的交互方式
 - 需要实现 intel 8042 PS2 控制器驱动
 - 在此基础上进一步实现键盘，鼠标驱动

适配驱动

- VGA 图形驱动（乔一凡）
 - 之前的 VGA 驱动都工作在字符模式下，我们要实现在图形模式下 VGA 的工作
 - 特殊寄存器的设置，显存的设置可以查阅 VGA 编程手册
- 网卡驱动
 - 已有 Tifflin kernel 中有特定的 rtl 8139 网卡驱动，也有有关 intel 网卡驱动的一系列实现
 - 我们将在真机上跑最后的 os，因此也会根据具体的网卡型号进行实现

可加载内核模块

- 背景：系统的一部分功能的代码不载入到内存中，只保留接口。需要使用时将对应代码载入到内存中并修改系统对应指针，完成模块加载
- 优点：
 - 可以动态调整接口对应代码，更加灵活
 - 减小内核占用内存空间
- 已有工作参考：
 - 蓝昶学长 OS 专题训练设计：64位

可加载内核模块

- 实现内核可加载模块，需要
 - 建立内核符号表，并将系统内核符号导出到内核符号表中，以便模块调用系统符号（杨国炜）
 - 将内核模块从文件读入内存（杨国炜）
 - 解析 **ELF** 模块，将符号，变量加入符号表（乔一凡）
 - 重定位，保证模块中能够调用系统符号（乔一凡）
 - 操作系统层面提供载入，注册模块的接口（杨国炜）

可加载内核模块

- 内核符号表采用 **hash** 表进行维护，同时在导入内核符号时可以过滤部分符号，提高内核安全性
- **ELF** 解析器：
 - 需要完成解析和符号重定位的功能，实现比较复杂
 - 参考蓝昶学长的实现
 - 注意要记录模块加载和卸载初始化的函数地址

可加载内核模块

- 重定位：
 - 根据 ELF 文件中的 Relocation Table 的指示完成重定位（内核符号，普通符号）
 - 平台相关性很大，处理起来比较复杂（R_X86_64_64, R_X86_64_32, R_X86_64_32S）分别处理

```
typedef struct {  
    Elf32_Addr    r_offset;  
    Elf32_Word    r_info;  
    Elf32_Sword   r_addend;  
} Elf32_Rela;
```

Field	Description
r_offset	要修改的地址值在相对于 section 起始地址的相对位置
r_info >> 8	符号表中符号的索引，这个符号在 r_offset 被引用
r_info & 255	重定位类型
r_addend	常数，计算新地址会用到

驱动模块化

- 需要实现 **LKM** 两个最基本的函数：（杨国炜，乔一凡，按之前驱动分工）
 - `init_module()`：需要注册驱动中的符号，供内核使用
 - `cleanup_module()`：解注册符号
- 维护系统当前所有的设备的列表，每项代表一个设备，驱动模块需要在加载/删除时相应注册/注销设备

谢谢大家！