Rustable

Rust 实现的 Raspberry Pi 3 OS

乔逸凡 2015013188
谭咏霖 2015011491

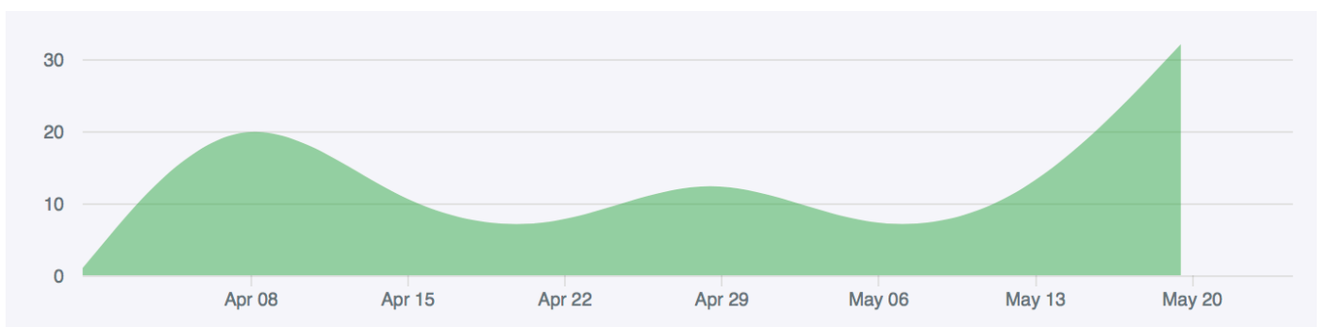
```
% find . -name "*.rs" | xargs wc -l
      10 ./build.rs
     176 ./src/kmain.rs
     473 ./src/shell.rs
     120 ./src/traps/syndrome.rs
      15 ./src/traps/trap_frame.rs
      19 ./src/traps/irq.rs
     104 ./src/traps/mod.rs
      21 ./src/traps/syscall/wait.rs
      48 ./src/traps/syscall/fork.rs
      45 ./src/traps/syscall/mod.rs
      22 ./src/traps/syscall/exit.rs
      22 ./src/traps/syscall/sleep.rs
      92 ./src/console.rs
      66 ./src/mutex.rs
     141 ./src/allocator/page.rs
      45 ./src/allocator/util.rs
     243 ./src/allocator/linked_list.rs
     449 ./src/allocator/first_fit.rs
     107 ./src/allocator/mod.rs
     339 ./src/allocator/tests.rs
       4 ./src/mm/mod.rs
       4 ./src/mm/vm/memory_manager.rs
      22 ./src/mm/vm/swap/mod.rs
      55 ./src/mm/vm/address.rs
     149 ./src/mm/vm/mod.rs
      34 ./src/mm/vm/page_fault.rs
     200 ./src/mm/pmm/mod.rs
      71 ./src/lang_items.rs
      77 ./src/fs/mod.rs
     105 ./src/fs/sd.rs
      13 ./src/process/mod.rs
      36 ./src/process/state.rs
     312 ./src/process/scheduler.rs
      32 ./src/process/elf.rs
      70 ./src/process/stack.rs
      37 ./src/process/syscall.rs
     231 ./src/process/process/mod.rs
      26 ./src/process/process/utils.rs
      98 ./src/aarch64.rs
    4133 total
```

111 commits

LegaDyan / Rustable build passing

Current Branches Build History Pull Requests More options

✓ master 乔逸凡	update travis yml.	#9 passed eedda63	7 min 51 sec about 2 hours ago



Rustable

Lab0
环境配置

Lab1
bootloader、启动

Lab2
物理内存管理

Lab3
虚拟内存管理

Lab4
进程控制块

Lab5
用户进程管理

Lab6
进程调度

Lab7
同步互斥

Lab8
文件系统

Lab0
环境配置

Lab1
bootloader、启动

Lab2
物理内存管理

Lab3
虚拟内存管理

Lab4
进程控制块

Lab5
用户进程管理

Lab6
进程调度

Lab7
同步互斥

Lab8
文件系统

Lab0

环境配置

Lab1

bootloader、启动

Lab7

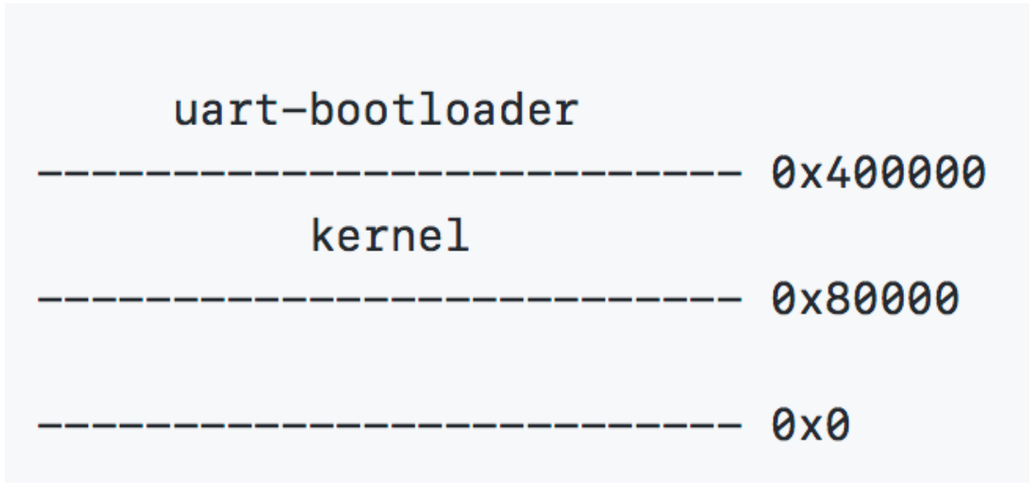
同步互斥

Lab8

文件系统

Lab1

bootloader、启动



- bootloader：bin 已实现；
- （伪）“bootloader”：通过 UART 读串口，将 kernel.img 写到内存 0x80000，并跳入，执行kernel。

Lab1

bootloader、启动

- bootloader：bin 已实现；
- （伪）“bootloader”：通过 UART 读串口，将 kernel.img 写到内存 0x80000，并跳入，执行kernel。
  - ❖ 填写页表（函数 vm_init）；
  - ❖ 使能 MMU（汇编实现）；
  - ❖ kernel 地址改为 0xFFFFFF0000800000。

Lab0

环境配置

Lab1

bootloader、启动

Lab7

同步互斥

Lab8

文件系统

Lab7

同步互斥

```rust
pub struct Allocator(Mutex<Option<imp::Allocator>>);


impl Allocator {
    pub const fn uninitialized() -> Self {
        Allocator(Mutex::new(None))
    }
    pub fn initialize(&self) {
        *self.0.lock() = Some(imp::Allocator::new());
    }
    fn init_memmap(&mut self, begin: usize) {
        self.0.lock().as_mut().expect("").alloc(begin);
    }
}


pub static ALLOCATOR: Allocator = Allocator::uninitialized();
ALLOCATOR.initialize();
ALLOCATOR.init_memmap(base);
```

Lab0

环境配置

Lab1

bootloader、启动

Lab7

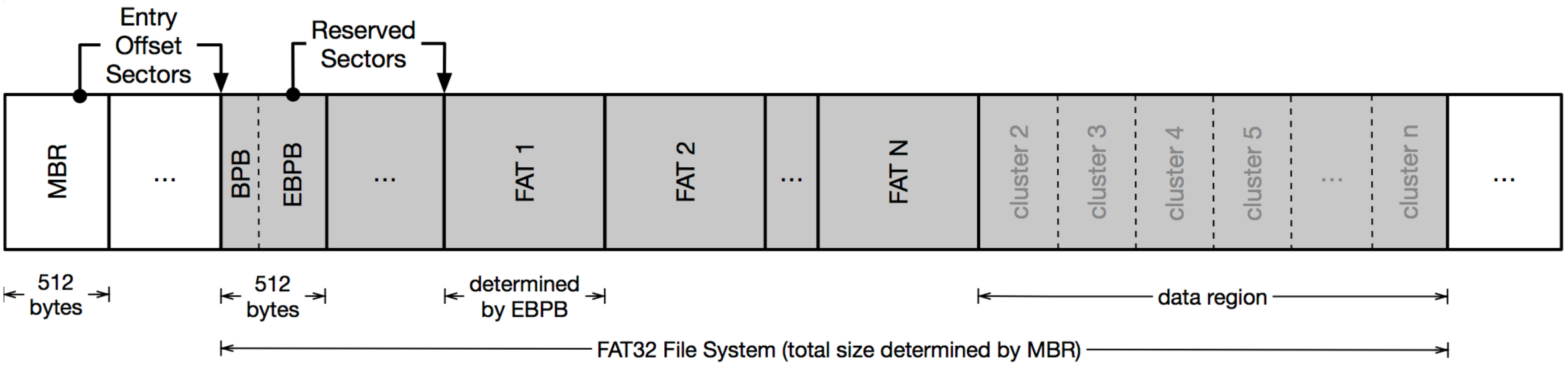
同步互斥

Lab8

文件系统

Lab8

文件系统


Entry
Offset
Sectors
Reserved
Sectors
MBR
...
BPB
EBPB
...
FAT 1
FAT 2
...
FAT N
cluster 2
cluster 3
cluster 4
cluster 5
...
cluster n
...
512
bytes
512
bytes
determined
by EBPB
data region
FAT32 File System (total size determined by MBR)

Lab0

环境配置

Lab1

bootloader、启动

Lab7

同步互斥

Lab8

文件系统

Lab0

环境配置

Lab1

bootloader、启动

Lab7

同步互斥

Lab8

文件系统

Lab2

物理内存管理

Lab3

虚拟内存管理

Lab4

进程控制块

Lab5

用户进程管理

Lab6

进程调度

内存

进程

Lab2

物理内存管理

Lab4

进程控制块

Lab5

用户进程管理

Lab3

虚拟内存管理

Lab6

进程调度

Lab2

物理内存管理

- 物理内存管理结构体 Pmm；
  - 探测物理内存大小和布局；
    - ATAG 數組
  - 页级管理的初始化 page_init
    - 通过遍历 Atag 数组获取连续的物理内存块
    - 预估出管理页级物理内存空间所需的 Page 结构的内存空间所需的内存大小。
    - 使用 ALLOCATOR 來管理保存连续空闲内存页。

Lab2

物理内存管理

- Allocator：页级物理内存管理的结构体
  - **管理空闲页（init_memmap)**
  - 管理用户页（init_user）
  - **分配页和释放页（alloc, dealloc）**
  - 清理进程使用的页（clear_page）
  - 拷贝页表和页（copy_page）
  - 分配指定虚疑地址的虚拟页（alloc_at）。

内存

进程

Lab2

物理内存管理

Lab4

进程控制块

Lab5

用户进程管理

Lab3

虚拟内存管理

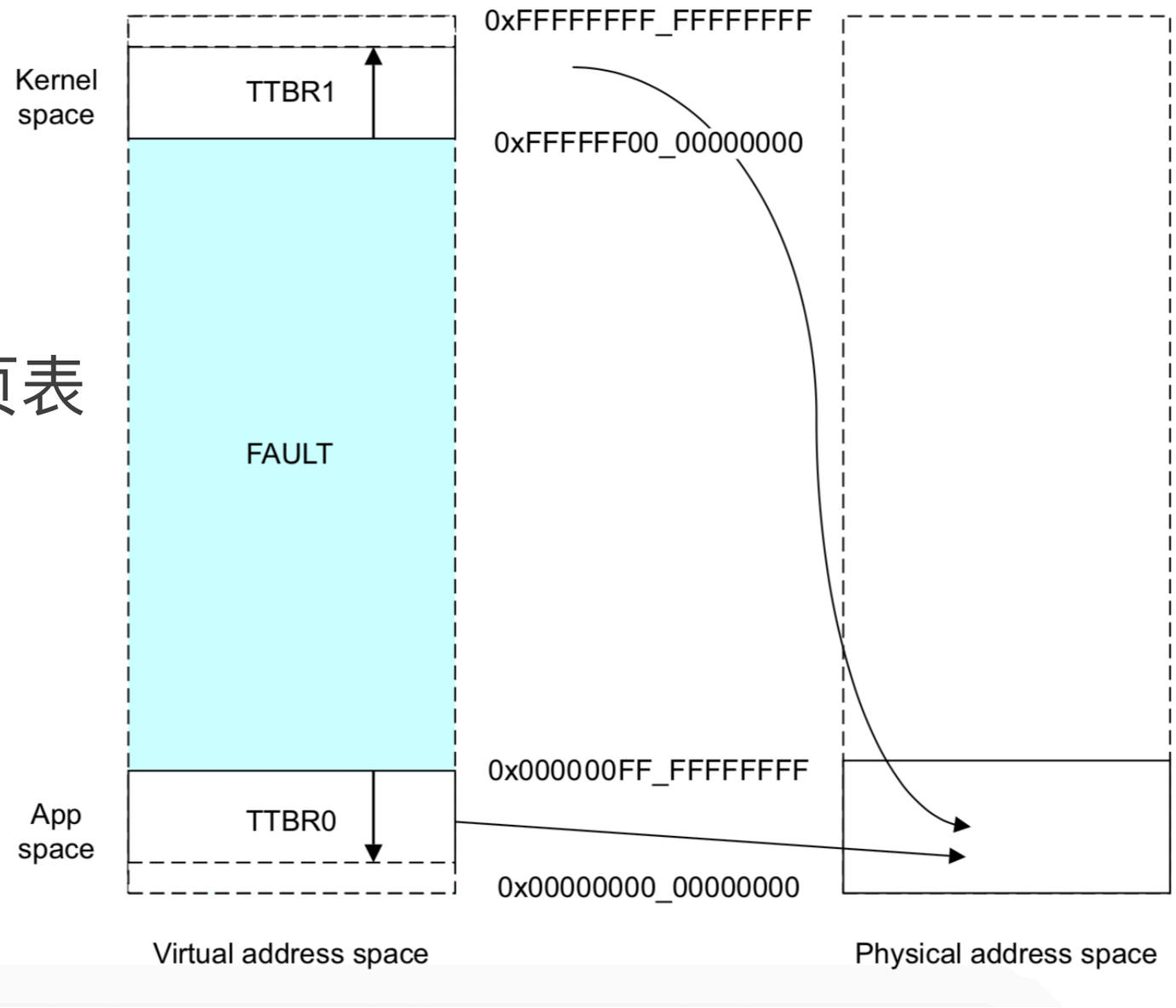
Lab6

进程调度

Lab3

虚拟内存管理

- 实现分页机制
  - 内核地址空间和
    用户地址空间的隔离
    - TTBR0 和 TTBR1 页表



Lab3

虚拟内存管理

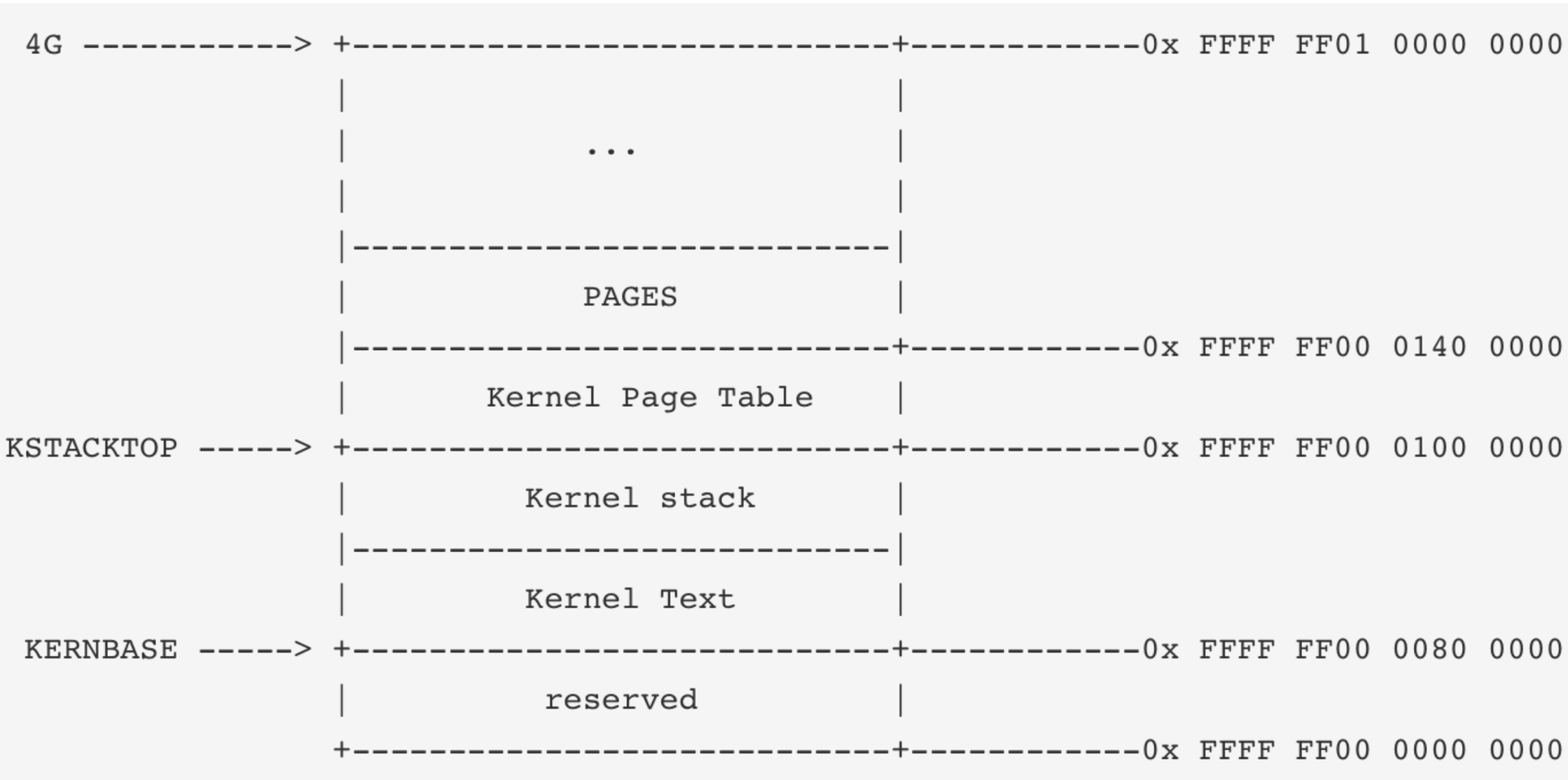
- 实现分页机制
  - 虚拟地址到物理地址的转换
    - 4 级页表
    - 物理页大小: 4KB

VA bits [47:39]	VA bits [38:30]	VA bits [29:21]	VA bits [20:12]	VA bits [11:0]
Level 0 Table Index Each entry contains: Pointer to L1 table (No block entry)	Level 1 Table Index Each entry contains: Pointer to L2 table Base address of 1GB block (IPA)	Level 2 Table Index Each entry contains: Pointer to L3 table Base address of 2MB block (IPA)	Level 3 Table Index Each entry contains: Base address off 4KB block (IPA)	Block offset and PA [11:0]

Lab3

虚拟内存管理

- 内核加载至高地址空间（0xFFFFFF0000800000）
- 在 bootloader 中建立高地址和低地址的线性映射



Lab3

虚拟内存管理

- 内核加载至高地址空间（0xFFFFFF0000800000）
- 在 bootloader 中建立高地址和低地址的线性映射
  - 切换完特权级 EL1 后：
  - 填写页表
  - 开启 MMU
    - 设置 TTBR0 和 TTBR1 寄存器
    - 设置 TCR_EL1 寄存器
    - 设置 SCTLR_EL1 寄存器

内存

进程

Lab2

物理内存管理

Lab3

虚拟内存管理

Lab4

进程控制块

Lab5

用户进程管理

Lab6

进程调度

Lab4

进程控制块

- 进程的状态：
  - Waiting (fn)
  - Running
  - Ready
  - Zombie
  - Wait_Proc (u32)
- TrapFrame
  - 页表地址（ttbr0）
  - pid

Lab4

进程控制块

- 为用户分配一个 Allocator 对象；
  - 初始化为固定大小（512M）；
    - Page 数组占 768 页，共 768 * 4k = 3M 额外开销；
- 在实际 alloc 时：
  - 在用户的 Allocator 上分配；
  - 访存时可能产生缺页，在 do_pgfault() 中 alloc() 物理页；
- 进程切换时更换全局 ALLOCATOR 包含的 Allocator 。

Lab4

进程切换

```
=================== PAGE LIST ===================
n_free: 25286
page_pa: 7c76000 property 765 isUsed false
page_pa: 7f77000 property 1 isUsed false
page_pa: 7c6f000 property 1 isUsed false
page_pa: 7973000 property 758 isUsed false
page_pa: 7f7a000 property 134 isUsed false
page_pa: 7970000 property 1 isUsed false
page_pa: 45c8000 property 759 isUsed false
page_pa: 7c6d000 property 1 isUsed false
page_pa: 7965000 property 1 isUsed false
page_pa: 7666000 property 1 isUsed false
page_pa: 735e000 property 759 isUsed false
page_pa: 766c000 property 755 isUsed false
page_pa: 7963000 property 1 isUsed false
page_pa: 765b000 property 1 isUsed false
...
page_pa: 14c2000 property 1 isUsed false
page_pa: 2755000 property 1 isUsed false
page_pa: 244d000 property 1 isUsed false
page_pa: 214e000 property 1 isUsed false
page_pa: 1e46000 property 759 isUsed false
page_pa: 244b000 property 1 isUsed false
page_pa: 2143000 property 1 isUsed false
page_pa: 1e44000 property 1 isUsed false
page_pa: 1b3c000 property 759 isUsed false
page_pa: 2141000 property 1 isUsed false
page_pa: 1e39000 property 1 isUsed false
page_pa: 1b3a000 property 1 isUsed false
page_pa: 1822000 property 1 isUsed false
page_pa: 1e37000 property 1 isUsed false
page_pa: 1b30000 property 1 isUsed false
page_pa: 14cb000 property 4 isUsed false
================================================
```

内存

进程

Lab2

物理内存管理

Lab4

进程控制块

Lab5

用户进程管理

Lab3

虚拟内存管理

Lab6

进程调度

Lab5

用户进程管理

- 调用 load_icode() 新建进程；
- fork()
  - 从 SCHEDULER 中拿出当前的 process；
  - 对 process 进行拷贝：
    - 新建 Allocator
    - 使用 copy_page() 拷贝物理页，并复制页表、Allocator；
  - 将拷贝的进程放入队尾，将被拷贝的进程放入队首（继续执行）；
- exit()
  - clear_page() 对物理页进行 dealloc
  - 将进程状态设为 Zombie

Lab5

用户进程管理

- load_icode()
- copy_page()
- clear_page()

Lab5

用户进程管理

- load_icode()
  - 创建进程的页表
  - **为用户的 allocator 初始化**
    - 分配一个 Page 数组（768页）
  - 读取 ELF 文件、读取 ProgramHeader
  - 建立用户栈空间
  - 設置 trapframe

Lab5

用户进程管理

- copy_page()
  - 拷贝 Page 数组
  - 拷贝源进程中使用了的页中保存数据
    - 同时为目标进程分配物理页和建立页表。
- clear_page()
  - 清理用户进程所用的空间
  - 释放分配过的页
  - 释放存放Page结构体的数组所用的空间

内存

进程

Lab2

物理内存管理

Lab4

进程控制块

Lab5

用户进程管理

Lab3

虚拟内存管理

Lab6

进程调度

Lab6

进程调度

- 切换进程：
  - 切换用户 Allocator 为内核 Allocator；
  - context_save 将 tf 保存在栈上；
  - 中断处理函数接受 sp（tf 指针）并传递；
  - SCHEDULER 直接对 tf 指针指向的内容进行修改；
  - eret；
- wait()
  - 参数为等待的进程 pid
  - 遍历进程队列，若等待进程状态为 Zombie 则可被调度

内存

进程

Lab2

物理内存管理

Lab4

进程控制块

Lab5

用户进程管理

Lab3

虚拟内存管理

Lab6

进程调度

Lab0
环境配置

Lab1
bootloader、启动

Lab2
物理内存管理

Lab3
虚拟内存管理

Lab4
进程控制块

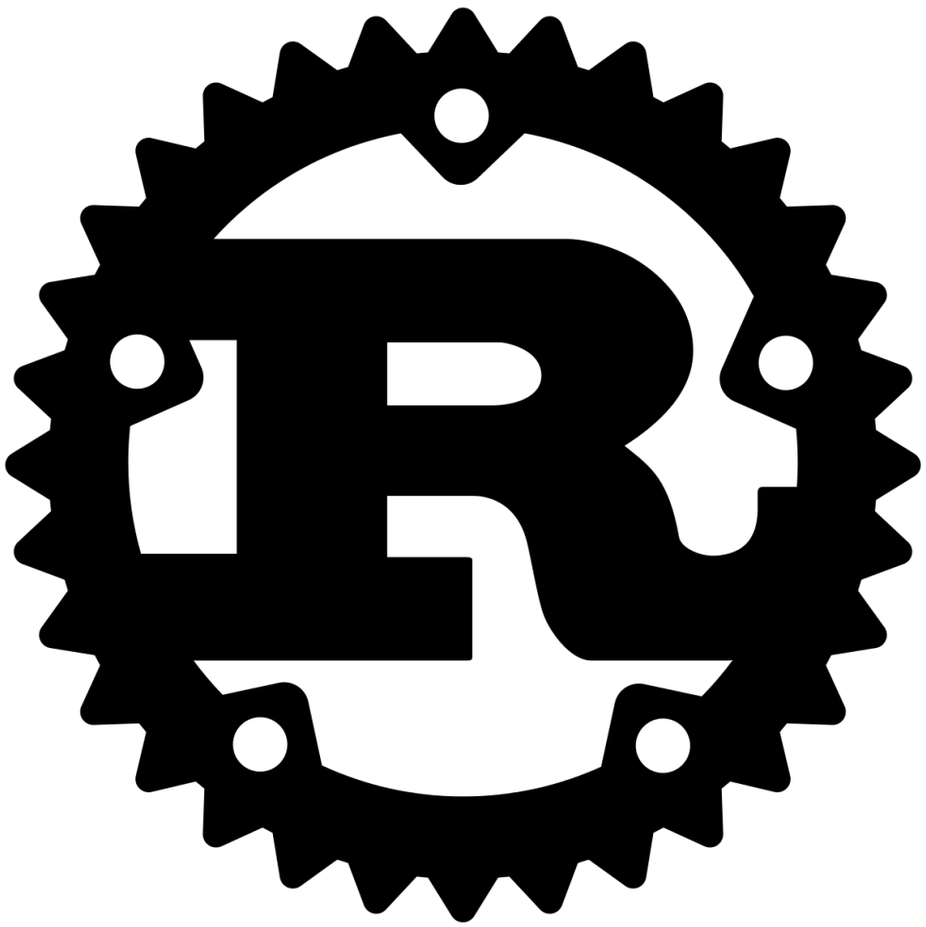
Lab5
用户进程管理

Lab6
进程调度

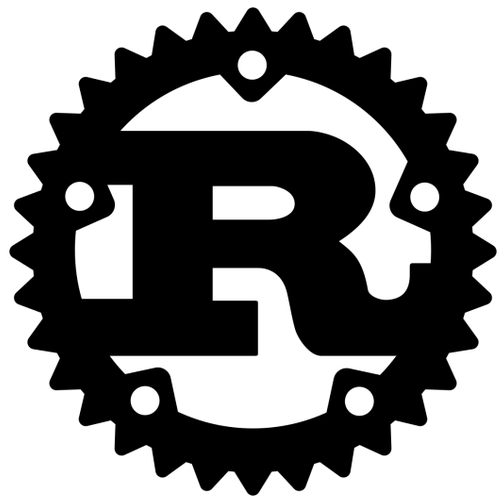
Lab7
同步互斥

Lab8
文件系统

Rustable



Pros



- 线程安全：Mutex
- 所有权：
  - 无需 free
  - 变量值安全
- 智能指针
  - Box
  - Rc

Cons

- 学习曲线陡峭
- 与编译器作斗争
  - 所有权
  - 生命周期
- 大量 unsafe
  - 野指针访问
  - mut static

Rustable

Demo

Rustable

Fin