Lab 4: RISC-V 虚拟内存管理

1. 实验简介

- 结合课堂学习的页式内存管理以及虚拟内存的相关知识，尝试在已有的程序上开启 MMU 并实现页映射，保证之前的进程调度能在虚拟内存下正常运行

2. 实验环境

- 在实验 3 的基础上进行（Docker Image）

3. 背景知识

3.1 虚拟内存

MMU（Memory Management Unit），负责`虚拟地址`到`物理地址`的转换。程序在cpu上运行时，他使用的虚拟地址会由MMU进行翻译。为了加速地址翻译的过程，现代cpu都引入了TLB（Translation Lookaside Buffer）。

分页机制的基本思想是将程序的虚拟地址空间划分为连续的，等长的虚拟页。`虚拟页和物理页的页长固定且相等`（一般情况下为4kb），从而操作系统可以方便的为每个程序构造页表，即虚拟页到物理页的映射关系。

逻辑上，该机制下的虚拟地址有两个部分组成：`1.虚拟页号`；`2.页内偏移`；在具体的翻译过程中，MMU首先解析得到虚拟地址中的虚拟页号，并通过虚拟页号查找到对应的物理页，最终用该物理页的起始地址加上页内偏移得到最终的物理地址。

3.2 RISC-V Virtual-Memory System (Sv39)

3.2.1 RISC-V `satp` Register（Supervisor Address Translation and Protection Register）

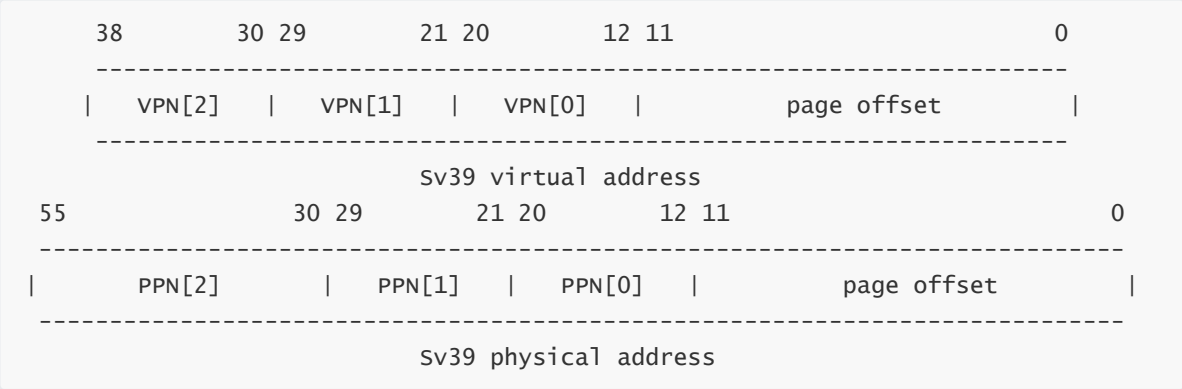
```
 63      60 59                  44 43                                0
 ------------------------------------------------------------------------
|   MODE   |         ASID         |                PPN                 |
 ------------------------------------------------------------------------
```

- MODE 字段的取值如下图：

```
                          RV 64
     ------------------------------------------------------------
    |  Value  |  Name  |  Description                            |
    |---------|--------|-----------------------------------------|
    |    0    | Bare   | No translation or protection            |
    |  1 - 7  | ---    | Reserved for standard use               |
    |    8    | Sv39   | Page-based 39 bit virtual addressing    | <-- 我们使用的
mode
    |    9    | Sv48   | Page-based 48 bit virtual addressing    |
    |    10   | Sv57   | Page-based 57 bit virtual addressing    |
    |    11   | Sv64   | Page-based 64 bit virtual addressing    |
    | 12 - 13 | ---    | Reserved for standard use               |
    | 14 - 15 | ---    | Reserved for standard use               |
     ------------------------------------------------------------
```

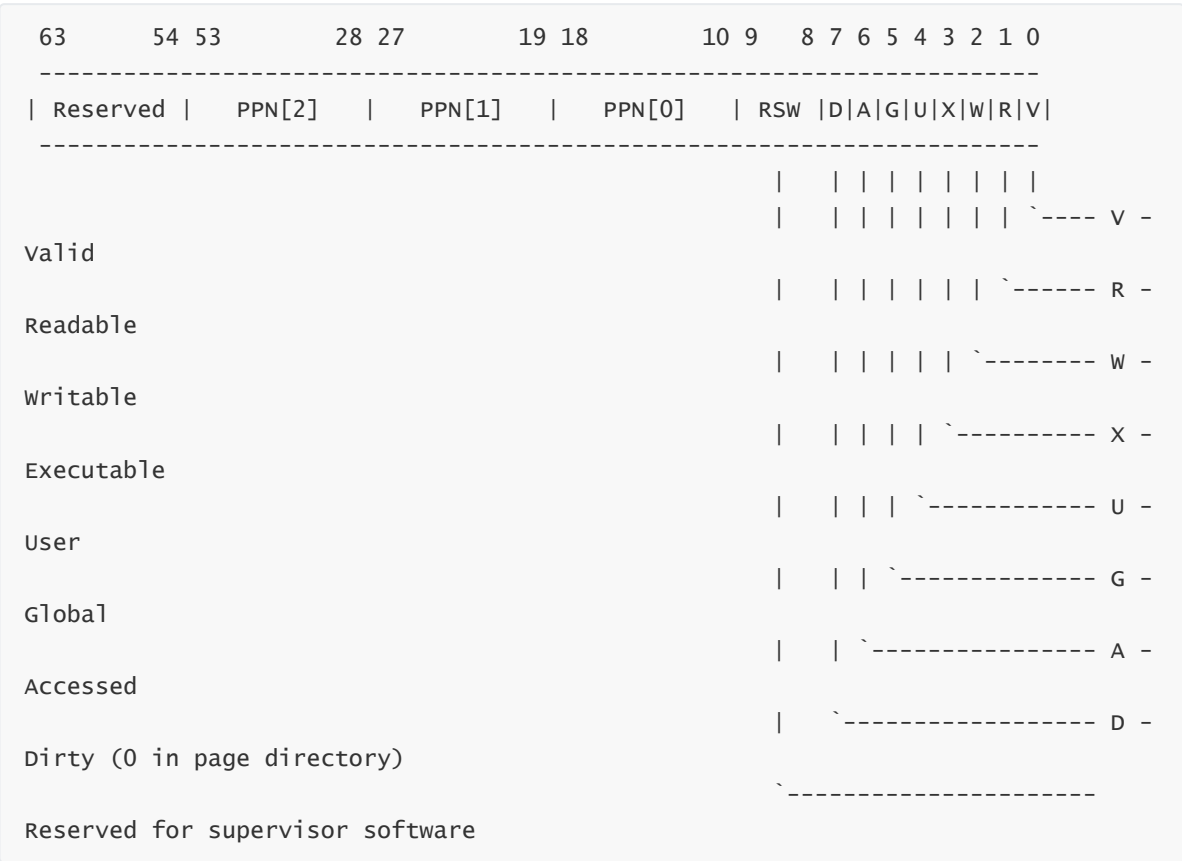
- ASID (Address Space Identifier)： 用来区分不同的地址空间，此次实验中直接置0即可。
- PPN (Physical Page Number)：顶级页表的物理页号，通常 `PPN = physical address >> 12`。

3.2.2 RISC-V Sv39 Virtual Address and Physical Address



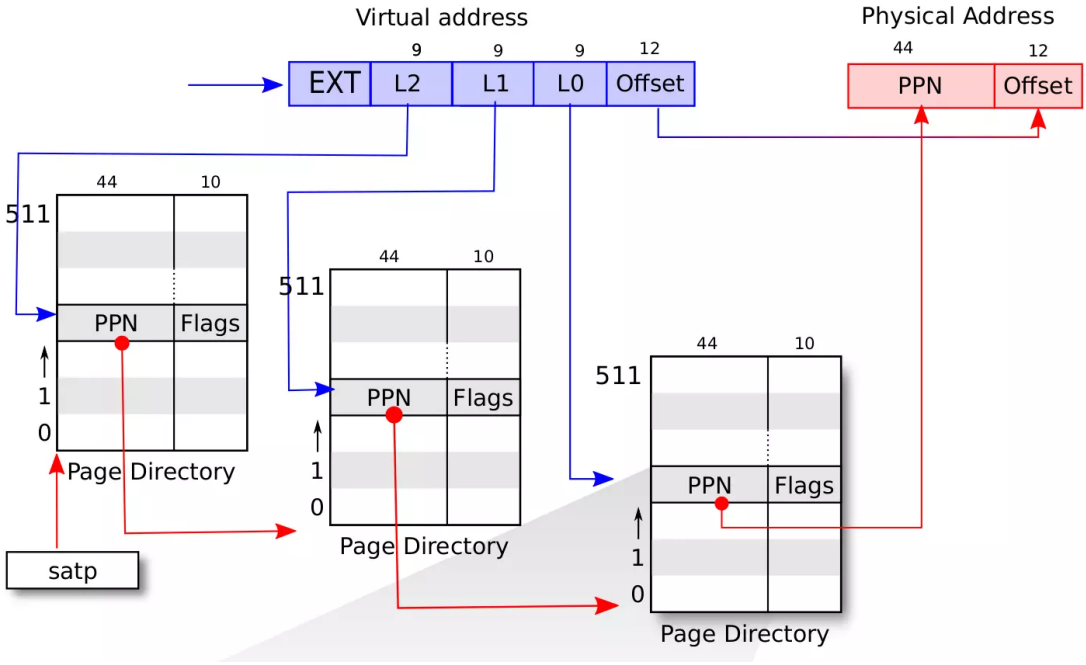
Sv39模式定义物理地址有56位，虚拟地址有64位。但是，虚拟地址的64位只有39位有效，63-39位在本次实验中（高地址映射）需要为1保证地址有效。Sv39支持三级页表结构，VPN2-0分别代表每级页表的`虚拟页号`，PPN2-0分别代表每级页表的`物理页号`。物理地址和虚拟地址的低12位表示页内偏移（page offset）。

3.2.3 RISC-V Sv39 Page Table Entry



- 0 ~ 9 bit: protection bits
  - V: 有效位，当 V = 0, 访问该PTE会产生Pagefault。
  - R: R = 1 该页可读。
  - W: W = 1 该页可写。
  - X: X = 1 该页可执行。
  - U,G,A,D,RSW本次实验中设置为0即可。

3.2.4 RISC-V Address Translation Details



- 1.从satp的 `PPN` 中获取根页表的物理地址。
- 2.通过pagetable中的VPN段,获取PTE。(可以把pagetable看成一个数组，VPN看成下标。PAGE_SIZE为4KB，PTE为64bit(8B), 所以一页中有4KB/8B=512个PTE，而每级VPN刚好有9位，与512个PTE一一对应)。
- 3.检查PTE的 `V bit`，如果不合法，应该产生page fault异常。
- 4.检查PTE的 `RWX` bits,如果全部为0，则从PTE中的PPN[2-0]得到的是下一级页表的物理地址，则回到第二步。否则当前为最后一级页表，PPN[2-0]得到的是最终物理页的地址。
- 5.将得到最终的物理页地址，与偏移地址相加，得到最终的物理地址。
  (以上为简要的地址翻译过程，完整过程参考这里)

4. 实验步骤

4.1 环境搭建

4.1.1 建立映射

同lab3的文件夹映射方法，目录名为lab4。

4.1.2 组织文件结构

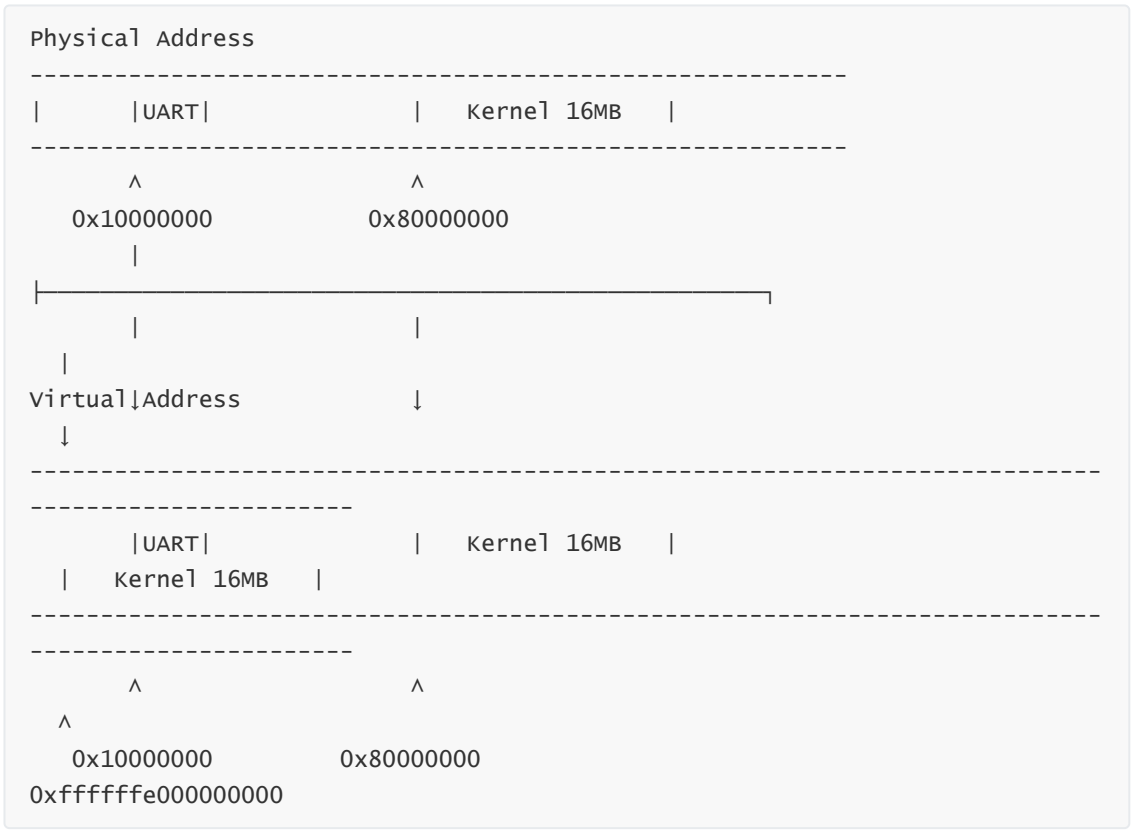
```
lab4
├── arch
│   └── riscv
│       ├── include
│       │   ├── put.h
│       │   ├── sched.h
│       │   └── vm.h
│       ├── kernel
│       │   ├── entry.S
│       │   ├── head.S
│       │   ├── Makefile
│       │   ├── sched.c
│       │   ├── strap.c
│       │   ├── vm.c
│       │   └── vmlinux.lds
```

```
|       └── Makefile
├── include
|   ├── put.h
|   ├── rand.h
|   └── test.h
├── init
|   ├── main.c
|   ├── Makefile
|   └── test.c
├── lib
|   ├── Makefile
|   ├── put.c
|   └── rand.c
└── Makefile
```

其中`vmlinux.lds`会提供给同学们。

4.2 创建映射

- 本次实验使用Sv39分配方案，支持3级页表映射
- 在`vm.c`中编写函数`create_mapping(uint64 *pgtbl, uint64 va, uint64 pa, uint64 sz, int perm)`，用作页表映射的统一接口，其中参数作用如下：
  - `pgtbl`为根页表的基地址
  - `va`，`pa`分别为需要映射的虚拟、物理地址的基地址
  - `sz`为映射的大小
  - `perm`为映射的读写权限
  - 函数中，若需要为页表空间分配物理页，可以自由管理分配`end`之后的物理内存
- 在`vm.c`中编写`paging_init`函数，调用`create_mapping`函数将内核起始（0x80000000）的16MB空间映射到高地址（以`0xffffffe000000000`为起始地址），同时也进行等值映射。将必要的硬件地址（如UART）进行等值映射，无偏移
- 映射图如下：



4.3 修改head.S

4.3.1 修改系统启动部分代码

- 在`_start`开头先设置`satp`寄存器为0，暂时关闭MMU
- 进入S模式后，在适当位置调用`paging_init`函数进行映射
- 设置

  ```
  satp
  ```

  的值以打开MMU
  - 注意`satp`中的PPN字段以4KB为单位
- 设置`stvec`为异常处理函数`trap_s`在**虚拟地址空间下**的地址
- 设置`sp`的值为虚拟地址空间下的`init_stack_top`
- 跳转到虚拟地址下的

  ```
  start_kernel
  ```

  ，并在虚拟地址空间中执行后续语句与进程调度
  - 可以先将`start_kernel`的虚拟地址装载在寄存器中，并使用`jr`指令进行跳转

4.3.2 修改M模式下异常处理代码

- 由于M模式下依然使用物理地址，使用虚拟地址将导致内存访问错误。因此，需要保留一片物理地址区域用于异常处理前保存所有寄存器的值
- `mscratch`寄存器是M mode下专用的临时寄存器。通常，它就用于保存M mode下上下文物理空间的地址。lds文件中分配出了1个page的空间用于储存进程上下文，其顶部标记为`stack_top`，请在head.S进入S mode之前的适当位置，将`mscratch`寄存器设置为`stack_top`的物理地址。
- 在M mode异常处理函数`trap_m`的开头，将`mscratch`与`sp`寄存器的值交换（hint: 使用`csrrw`指令），使用上下文空间作为`trap_m`的栈并保存`x1-x31`寄存器
- 在`trap_m`返回前将`mscratch`与`sp`寄存器的值重新交换回来

4.4 修改进程调度相关代码sched.c

4.4.1 修改task_init()调整为虚拟地址

- 由于开启了MMU，因此我们需要修改进程相关代码，确保将`task_struct`以及各进程的地址划分到虚拟地址空间。

4.4.2 在进程调度时打印task_struct地址

- 修改`schedule()`函数在调度时的打印输出，要求打印出`current`和`task[next]`的地址以及进程栈顶`sp`的值

4.5 完成对不同section的保护

- 通过修改调用

  ```
  create_mapping
  ```

  时的

```
perm
```

参数，修改对不同section所在页属性的设置，完成对不同section的保护

- 包括：text r-x, rodata r--, other rw-
- **思考题：如何验证这些属性是否成功被保护**

- 在`head.S`中，通过修改`medeleg`寄存器，将instruction/load/store page fault托管到S模式下
- 修改`strap.c`中的handler，添加对page fault的打印

4.6 编译及测试

```
ZJU OS LAB 4             GROUP-XX
[PID = 1] Process Create Successfully! counter = 7 priority = 5
[PID = 2] Process Create Successfully! counter = 6 priority = 5
[PID = 3] Process Create Successfully! counter = 5 priority = 5
[PID = 4] Process Create Successfully! counter = 4 priority = 5
[!] Switch from task 0 [task struct: 0xffffffe000ff2000, sp: 0xffffffe000ff3000]
to task 4 [task struct: 0xffffffe000fee000, sp: 0xffffffe000fef000], prio: 5,
counter: 4
tasks' priority changed
[PID = 1] counter = 7 priority = 1
[PID = 2] counter = 6 priority = 4
[PID = 3] counter = 5 priority = 5
[PID = 4] counter = 4 priority = 4
[!] Switch from task 4 [task struct: 0xffffffe000fee000, sp: 0xffffffe000fef000]
to task 1 [task struct: 0xffffffe000ff1000, sp: 0xffffffe000ff2000], prio: 1,
counter: 7
tasks' priority changed
[PID = 1] counter = 7 priority = 5
[PID = 2] counter = 6 priority = 5
[PID = 3] counter = 5 priority = 5
[PID = 4] counter = 3 priority = 2
...
```

5. 实验任务与要求

请仔细阅读背景知识，理解如何建立页表映射，并按照实验步骤完成实验，撰写实验报告，需提交实验报告以及整个工程的压缩包。

- 实验报告：
  - 各实验步骤截图以及结果分析
  - 回答思考题
  - 实验结束后心得体会
  - 对实验指导的建议（可选）
- 工程文件
  - 所有source code（确保make clean）
- 最终目录
  - 将Lab4_319010XXXX目录压缩并打包（若分组，则一组只需要一位同学提交）

```
lab4_319010XXXX
├── arch
|   └── riscv
|       ├── include
```

```
|       |   ├── put.h
|       |   ├── sched.h
|       |   └── vm.h
|       ├── kernel
|       |   ├── entry.S
|       |   ├── head.S
|       |   ├── Makefile
|       |   ├── sched.c
|       |   ├── strap.c
|       |   ├── vm.c
|       |   └── vmlinux.lds
|       └── Makefile
├── include
|   ├── put.h
|   ├── rand.h
|   └── test.h
├── init
|   ├── main.c
|   ├── Makefile
|   └── test.c
├── lib
|   ├── Makefile
|   ├── put.c
|   └── rand.c
├── Makefile
└── report
    └── 319010XXXX.pdf
```

本文贡献者

> 王星宇（背景知识）
> 朱璟森，沈韬立（实验步骤）