Lab 3: RV64 虚拟内存管理

刘扬 3210105488

1 实验过程

本次实验需要实现虚拟地址到物理地址的映射，并对不同的段进行相应的权限设置。

1.1 准备工作

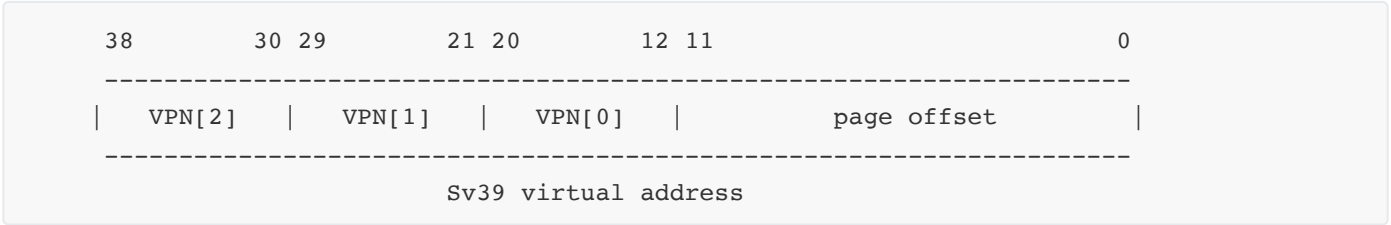
修改`defs.h`和`MakeFile`，替换`vmlinux.lds`，不赘述。

1.2 `setup_vm`的实现

将 0x80000000 开始的 1GB 区域进行两次映射，其中一次是等值映射 (PA == VA) ，另一次是将其映射至高地址 (PA + PV2VA_OFFSET == VA)。如下图所示：

在`setup_vm`中需要将 1GB 连续的物理地址映射到虚拟地址，这里不需要使用多级页表，只采用`PTE`的`PPN[2]`存储索引，这样物理地址的后 30 位都用作页内偏移，刚好能映射 1GB 的地址空间，根页表的每个 entry 都对应 1GB 的区域。

1.2.1 `early_pgtbl[]`的index计算



虚拟地址的如上所示，`VPN[2]`即是我们需要拿到的index，因此`index = (VA >> 30) & 0x1ff`

1.2.2 写`PTE`

```
 63      54 53        28 27        19 18        10 9   8 7 6 5 4 3 2 1 0
 -------------------------------------------------------------------------
| Reserved |   PPN[2]   |   PPN[1]   |   PPN[0]   | RSW |D|A|G|U|X|W|R|V|
 -------------------------------------------------------------------------
```

Page Table Entry 的权限 V | R | W | X 位设置为 1,其他权限位置 0，PPN对应物理地址的PPN：

```
 55                30 29        21 20        12 11                          0
 ------------------------------------------------------------------------------
|       PPN[2]       |   PPN[1]   |   PPN[0]   |          page offset          |
 ------------------------------------------------------------------------------
                            Sv39 physical address
```

即`PTE`的63 - 10位对应物理地址的55 - 12位，因此`early_pgtbl[index] = ((PA >> 12) << 10) | 0xf`

1.2.3 `setup_vm`代码

```c
void setup_vm(void) {
    memset(early_pgtbl, 0x0, PGSIZE);
    unsigned long PA = PHY_START;
    unsigned long VA1 = PA;
    unsigned long VA2 = PA + PA2VA_OFFSET;
    int index;
    index = (VA1 >> 30) & 0x1ff;
    early_pgtbl[index] = ((PA >> 12) << 10) | 0xf;
    index = (VA2 >> 30) & 0x1ff;
    early_pgtbl[index] = ((PA >> 12) << 10) | 0xf;
}
```

1.2.4 `relocate`实现

`relocate`用来设置`satp`，并且跳转到虚拟地址，分4步骤进行：

- 设置`ra`,`sp`

  将`ra`和`sp`加上偏移量即让它们指向对应的虚拟地址

  ```
  li t0, 0xffffffdf80000000
  add ra, ra, t0
  add sp, sp, t0
  ```

- 设置`sapt`

  `satp`寄存器如下所示：

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

因为本次实验是39位的虚拟地址，因此MODE字段置 8，ASID本次实验直接置 0，PPN == PA >> 12

```
li t0, 8
slli t0, t0, 60
la t1, early_pgtbl
li t2, 0xffffffdf80000000
sub t1, t1, t2
srli t1, t1, 12
or t0, t0, t1
csrw satp, t0
```

需要注意这里的`early_pgtbl`在符号表里已经是虚拟地址，因此需要手动减去偏移量。

- flush tbl
- flush icache

1.2.5 修改`_start`

在`_start`中调用`setup_vm`和`relocate`

```
_start:
    # set sp first for the call
    la sp, boot_stack_top # sp <- address of stack
    li t0, 0xffffffdf80000000
    sub sp, sp, t0
    call setup_vm
    call relocate
    # memory init
    call mm_init
    ...
```

需要注意的是这里的`sp`同样在符号表里已经指向虚拟地址，需要手动减去偏移量。

1.3 `setup_vm_final`的实现

1.3.1 修改`mm.c`

由于`setup_vm_final`中需要申请页面的接口，应该在其之前完成内存管理初始化，`mm.c`中初始化的函数接收的起始结束地址需要调整为虚拟地址。

```
kfreerange(_ekernel, (char *)(PHY_END+PA2VA_OFFSET));
```

1.3.2 `setup_vm_final()`实现

引入`vmlinux.lds`的`_stext`,`_srodata`,`_sdata`作为外部变量，这样便于在做映射时取地址。

```
extern char _stext[];
extern char _srodata[];
extern char _sdata[];
```

在`setup_vm_final`中需要做如下操作：

- 分别对`text`,`rodata`,`other memory`做映射，在做映射时用到了`create_mapping`函数，该函数的实现在1.3.3中给出。

  ```
  // mapping kernel text X|-|R|V
  create_mapping(swapper_pg_dir, (uint64)_stext, (uint64)_stext - PA2VA_OFFSET,
  _srodata - _stext, 11);

  // mapping kernel rodata -|-|R|V
  create_mapping(swapper_pg_dir, (uint64)_srodata, (uint64)_srodata - PA2VA_OFFSET,
  _sdata - _srodata, 3);

  // mapping other memory -|W|R|V
  create_mapping(swapper_pg_dir, (uint64)_sdata, (uint64)_sdata - PA2VA_OFFSET,
  PHY_SIZE - (_sdata - _stext), 7);
  ```

- 设置`satp`

  ```
  uint64 set_satp = (((uint64)(swapper_pg_dir) - PA2VA_OFFSET) >> 12) | ((uint64)8 <<
  60);
  csr_write(satp, set_satp);
  ```

- flush tlb
- flush icache

1.3.3 `create_mapping`实现

`create_mapping`函数中的每次循环映射一页 4KB 的物理地址到虚拟地址，具体流程为：

1. 将根页表的基地址作为`pgtbl`，根据`pgtbl`和`vpn`得到`pte`
2. 若 PTE 的有效位 V 为 0，则申请一块内存空间作为新的二级页表，并将新建的二级页表的地址存放到 PTE 中，并将有效位 V 置 1
3. 根据 PTE 的内容求出二级页表的物理地址，在二级页表中用同样的方法新建一个一级页表或求得一级页表的地址
4. 在一级页表中求得 PTE 的地址，将物理地址存入 PTE，将有效位 V 置 1，根据`perm`改写权限位

```
create_mapping(uint64 *pgtbl, uint64 va, uint64 pa, uint64 sz, uint64 perm) {
    uint64 cur_vpn;
    uint64* cur_pgtbl;
```

```
    uint64 cur_pte;
    uint64* new_pg;

    int i = sz / PGSIZE + 1;
    while (i--)
    {
        cur_pgtbl = pgtbl;
        cur_vpn = ((uint64)(va) >> 30) & 0x1ff;
        cur_pte = *(cur_pgtbl + cur_vpn);
        if (!(cur_pte & 1)) {
            new_pg = (uint64*)kalloc();
            cur_pte = ((((uint64)new_pg - PA2VA_OFFSET) >> 12) << 10) | 0x1;
            *(cur_pgtbl + cur_vpn) = cur_pte;
        }

        cur_pgtbl = (uint64*)((cur_pte >> 10) << 12);
        cur_vpn = ((uint64)(va) >> 21) & 0x1ff;
        cur_pte = *(cur_pgtbl + cur_vpn);
        if (!(cur_pte & 1)) {
            new_pg = (uint64*)kalloc();
            cur_pte = ((((uint64)new_pg - PA2VA_OFFSET) >> 12) << 10) | 0x1;
            *(cur_pgtbl + cur_vpn) = cur_pte;
        }

        cur_pgtbl = (uint64*)((cur_pte >> 10) << 12);
        cur_vpn = ((uint64)(va) >> 12) & 0x1ff;
        cur_pte = ((pa >> 12) << 10) | (perm & 0xf);
        *(cur_pgtbl + cur_vpn) = cur_pte;

        va += PGSIZE;
        pa += PGSIZE;
    }
    return;
}
```

1.3.4 修改`_start`

```
_start:
    ...
    call setup_vm
    call relocate
    call mm_init
    call setup_vm_final
    call task_init
    ...
```

1.4 编译及测试

结果如下：

```
Boot HART ID              : 0
Boot HART Domain          : root
Boot HART ISA             : rv64imafdcsu
Boot HART Features        : scounteren,mcounteren,time
Boot HART PMP Count       : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count      : 0
Boot HART MHPM Count      : 0
Boot HART MIDELEG         : 0x0000000000000222
Boot HART MEDELEG         : 0x000000000000b109
...mm_init done!
...proc_init done!
2022 Hello RISC-V
sstatus: 24578
sscratch: 5488
[S] Supervisor Mode Timer Interrupt
switch to [PID = 2 PRIORITY = 88 COUNTER = 9]
[PID = 2] is running. auto_inc_local_var = 1
[S] Supervisor Mode Timer Interrupt
[PID = 2] is running. auto_inc_local_var = 2
[S] Supervisor Mode Timer Interrupt
[PID = 2] is running. auto_inc_local_var = 3
[S] Supervisor Mode Timer Interrupt
[PID = 2] is running. auto_inc_local_var = 4
[S] Supervisor Mode Timer Interrupt
[PID = 2] is running. auto_inc_local_var = 5
QEMU: Terminated
```

2 思考题

1.验证 `.text`，`.rodata` 段的属性是否成功设置，给出截图。

`.text` 段具有执行和读权限。`.rodata` 段只具有读权限。

1. executable

程序可以执行说明 `.text` 段有执行权限。

程序在进 `start_kernel` 前先执行 `call _srodata`，并在在trap_handler处print scause，可以看到

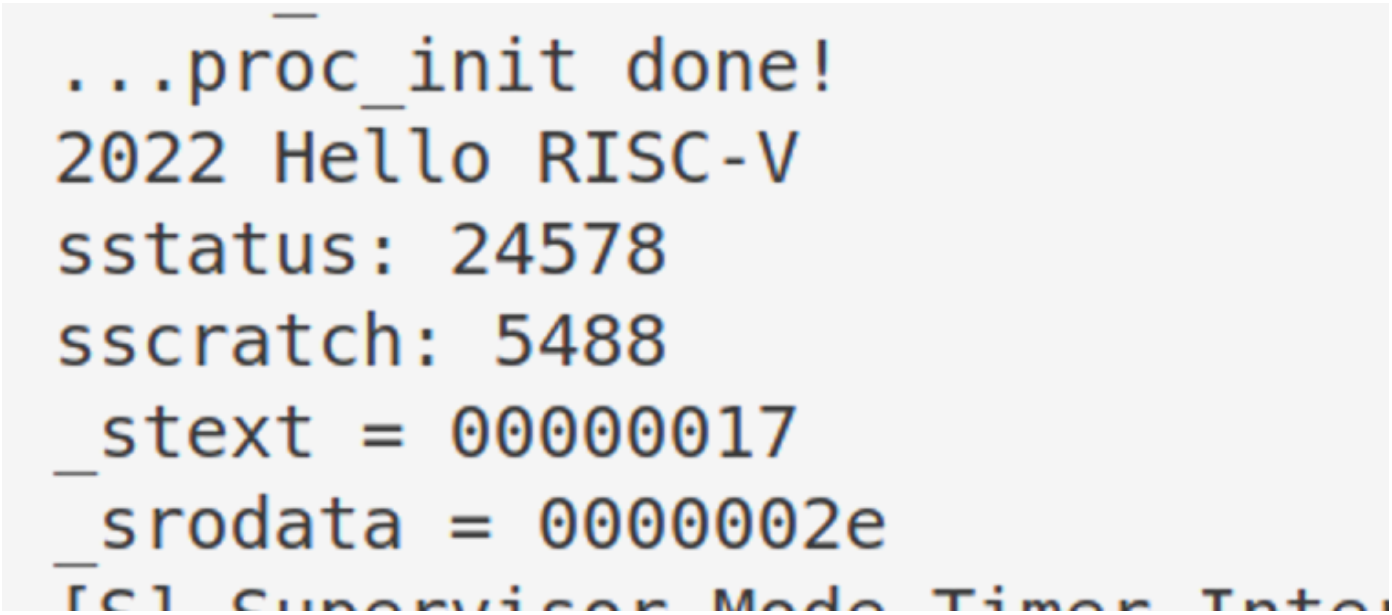
```
scause = 000000000000000c, sepc = ffffffe000202000
```

`scause` 为Instruction Page Fault，说明 `.rodata` 段不可执行（也就是不能从那里读取指令。

2. read

在`main.c`中添加如下代码

```
printk("_stext = %x\n", *_stext);
printk("_srodata = %x\n", *_srodata);
```

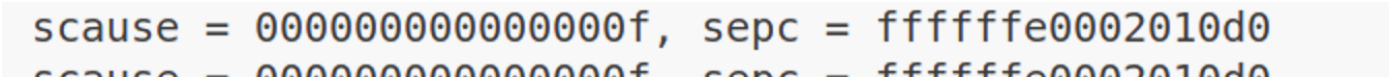


`_stext`和`_srodata`都具有读权限

3. write

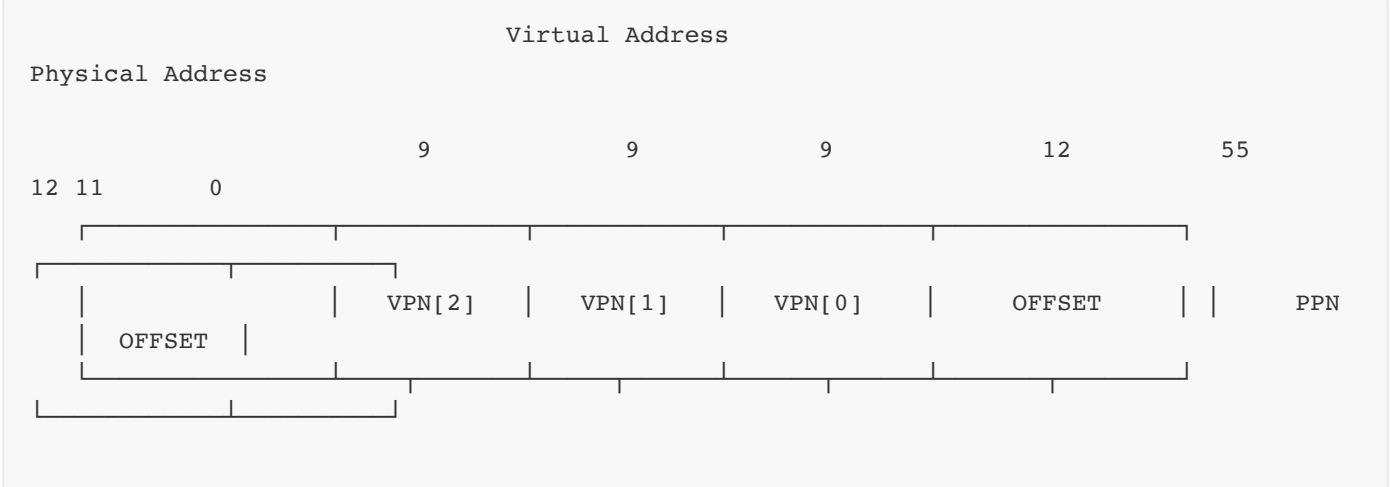
在`main.c`中分别对`_stext`和`_srodata`进行修改

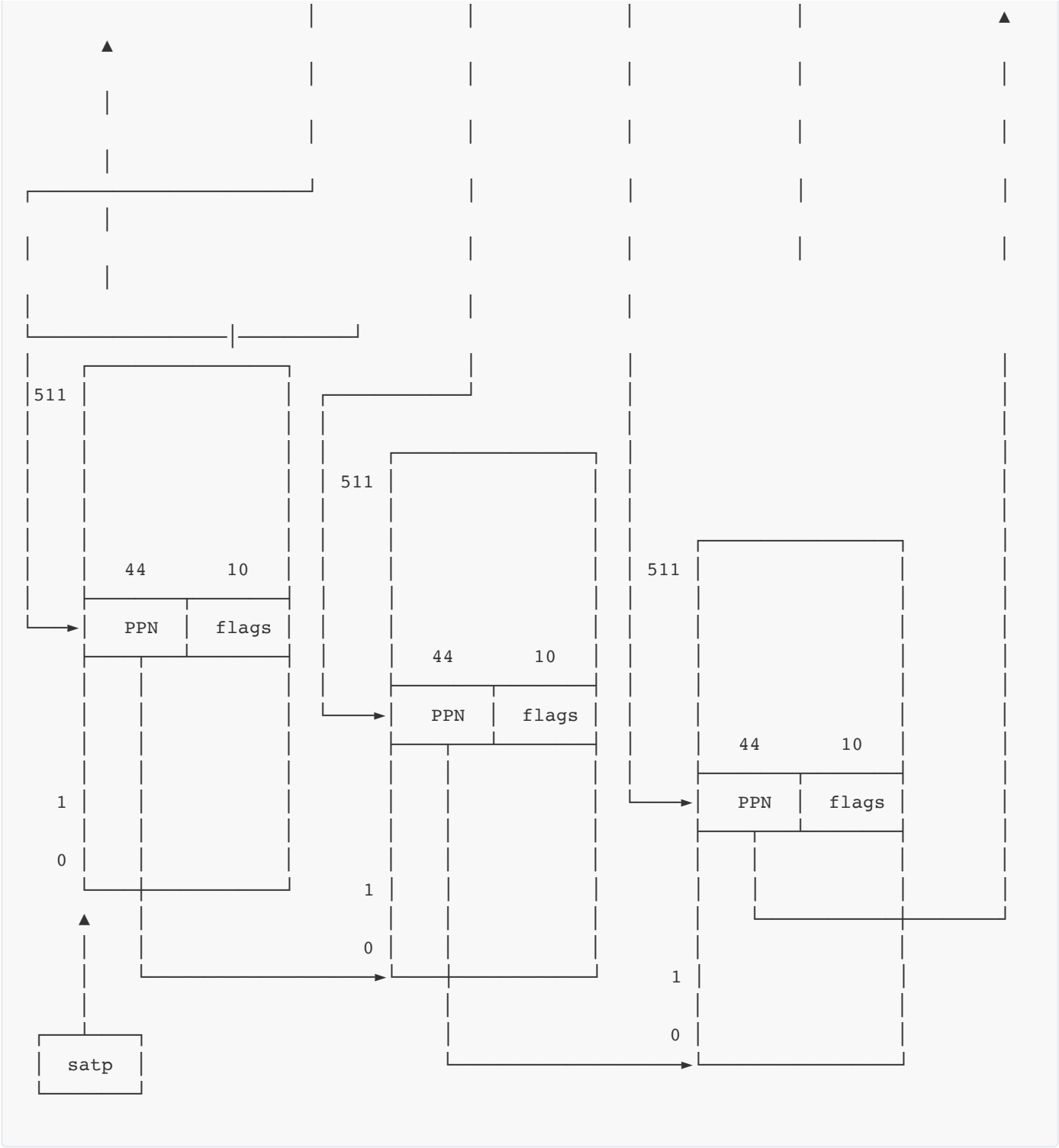
```
*_stext = 123;
//*_srodata = 321;
```



`scause`为Store/AMO Page Fault，说明 .text 和 .rodata 段都不可写。

2.为什么我们在 `setup_vm` 中需要做等值映射?





如图所示，在访问三级页表时，我们都是将VPN转换成PPN去拿到物理地址上的PTE。如果缺少等值映射，直接使用该物理地址将导致内存访问错误。

3.在 Linux 中，是不需要做等值映射的。请探索一下不在 `setup_vm` 中做等值映射的方法。

在做三级页表映射即`setup_vm_final`中，将PTE拿到的物理地址的值加上偏移量来作为下次访问的地址，即

```
cur_pgtbl = (uint64*)((cur_pte >> 10) << 12 + PA2VA_OFFSET)
```

3 实验心得

这次实验debug比较困难，因为gdb进去以后就是虚拟地址，需要手动在0x8020 0000打断点单步跟踪`_start`。需要注意`_start`中设置`sp`和`early_pgtbl`都应该物理地址，需要自己手动减去偏移量。