浙江大学实验报告

Lab 4：`RV64`虚拟内存管理

课程名称：　操作系统　实验类型：　综合

实验项目名称：　`RV64`虚拟内存管理

学生姓名：汪辉　专业：计算机科学与技术　学号：3190105609

同组学生姓名：个人实验　　指导老师：　季江民

电子邮件：3190105609@zju.edu.cn

实验地点：　曹西503　实验日期：　2021 年 12 月 23 日

组员：李瑞凝

分工说明：李瑞凝完成`head.S`修改和`vm.c`中`setup`函数；汪辉完成`create_mapping`映射函数修改及调试工作。实验报告两人各自完成。

1 实验目的

- 学习虚拟内存的相关知识，实现物理地址到虚拟地址的切换。
- 了解 RISC-V 架构中 SV39 分页模式，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。

2 实验内容

完善代码

本实验基于lab3内容进行，需要从repo中同步以下现成文件：

`vmlinux.lds.S`，`Makefile`

需要基于lab3工程修改以下文件内容：

1. `arch/riscv/kernel/head.S`

还需新建和编辑以下文件：

1. `arch/riscv/include/vm.h`
2. `arch/riscv/kernel/vm.c`

修改`defs.h`

按照实验指导，在lab4中引入了虚拟内存的映射，硬射中使用到一些自定义的宏，因此需要在`defs.h`添加以下内容：

```
#define OPENSBI_SIZE (0x200000)
#define VM_START (0xffffffe000000000)
#define VM_END (0xffffffff00000000)
#define VM_SIZE (VM_END - VM_START)
#define PA2VA_OFFSET (VM_START - PHY_START)
```

修改`head.S`

之前实验中的`head.S`初始化内存管理并启动start_kernel，由于lab4实现物理内存到虚拟内存的映射，需要在调用`mm_init`之前完成虚拟内存硬射。

```
    .section .text.init
    .globl _start
_start:
    la sp, boot_stack_top
    call setup_vm
    call relocate # in lab4
    call mm_init # in lab3
    call setup_vm_final # in lab4
    call task_init # in lab3
```

除此之外，`relocate`在`head.S`中实现，在`_start`调用`relocate`时跳转到该段运行。

```
    # lab4
relocate:
    # set ra = ra + PA2VA_OFFSET
    # set sp = sp + PA2VA_OFFSET (If you have set the sp before)
    ######################
    # YOUR CODE HERE #
    li t0, 0xffffffdf80000000
    add ra,ra,t0
    add sp, sp, t0
    ######################
    # set satp with early_pgtbl
    ######################
    # YOUR CODE HERE #
    li t0, 0x8000000000000000
    la t1, early_pgtbl
    srli t1, t1, 12
    add t0, t0, t1
    csrw satp, t0
    # flush tlb
    sfence.vma zero, zero
    ret
```

添加`vm.h`

```
#include "mm.h"
#include "defs.h"
#include "string.h"
// vm头文件包含设计的函数即可
void setup_vm(void);
void setup_vm_final(void);
void create_mapping(uint64* pgtbl,uint64 va, uint64 pa, uint64 sz, int perm);
```

添加`vm.c`

```c
#include "vm.h"
#include "printk.h"
// vmlinux.lds.S 中指定的各个段名
extern char _stext[];
extern char _etext[];
extern char _srodata[];
extern char _erodata[];
extern char _sdata[];
extern char _edata[];

/* 创建多级页表映射关系 */
void create_mapping(uint64 *pgtbl, uint64 va, uint64 pa, uint64 sz, int perm);
```

1. 初始化函数 `setup_vm`：

```c
/* early_pgtbl: 用于 setup_vm 进行 1GB 的 映射。 */
unsigned long early_pgtbl[512] __attribute__((__aligned__(0x1000)));
void setup_vm(void) {
    /*
    1. 由于是进行 1GB 的映射 这里不需要使用多级页表
    2. 将 va 的 64bit 作为如下划分： | high bit | 9 bit | 30 bit |
    high bit 可以忽略
    中间9 bit 作为 early_pgtbl 的 index
    低 30 bit 作为 页内偏移 这里注意到 30 = 9 + 9 + 12， 即我们只使用根页表， 根页表
的每个 entry 都对应 1GB 的区
    域。
    3. Page Table Entry 的权限 V | R | W | X 位设置为 1
    */
    early_pgtbl[2] = (0x80000000 >> 2) + 15;
    early_pgtbl[384] = (0x80000000 >> 2) + 15;
    // //the first mapping
    // early_pgtbl[2] = 0x80000;
    // early_pgtbl[2] = (uint64)(early_pgtbl[2] << 10);
    // early_pgtbl[2] = (uint64)(early_pgtbl[2] + 0xf);

    // //the secong mapping
    // early_pgtbl[384] = early_pgtbl[2];
    // return;
}
```

2. 初始化函数 `setup_vm_final`：

```c
/* swapper_pg_dir: kernel pagetable 根目录， 在 setup_vm_final 进行映射。*/
unsigned long swapper_pg_dir[512] __attribute__((__aligned__(0x1000)));

void setup_vm_final(void) {
    memset(swapper_pg_dir, 0x0, PGSIZE);
    // // No OpenSBI mapping required
    // mapping kernel text X|-|R|V
    create_mapping(swapper_pg_dir, _stext , _stext - PA2VA_OFFSET, _etext -
_stext, 11);
    // mapping kernel rodata -|-|R|V
    create_mapping(swapper_pg_dir, _srodata , _srodata - PA2VA_OFFSET,
_erodata - _srodata, 3);
    // mapping other memory -|W|R|V
```

```c
    create_mapping(swapper_pg_dir, _sdata , _sdata - PA2VA_OFFSET, PHY_END +
PA2VA_OFFSET - (uint64)_sdata, 7);
    // set satp with swapper_pg_dir
    uint64 csrstap = (((uint64)swapper_pg_dir - PA2VA_OFFSET) >> 12) +
0x8000000000000000;
    asm volatile("csrw satp, %0" :: "r"(csrstap));
    // YOUR CODE HERE
    // flush TLB
    asm volatile("sfence.vma zero, zero");
    // printk("create mapping is ok!\n") ;
    return;
}
```

3. 多级映射函数`create_mapping`:

```c
/* 创建多级页表映射关系 */
void create_mapping(uint64 *pgtbl, uint64 va, uint64 pa, uint64 sz, int
perm) {
    /*
    pgtbl 为根页表的基地址
    va, pa 为需要映射的虚拟地址、物理地址
    sz 为映射的大小
    perm 为映射的读写权限，可设置不同section所在页的属性，完成对不同section的保护
    创建多级页表的时候可以使用 kalloc() 来获取一页作为页表目录
    可以使用 V bit 来判断页表项是否存在
    */

    uint64 va_i, vpn0, vpn1, vpn2;
    int PG_NUM = (sz+4095)/4096 ; // ceil the page number
    for ( int i = 0 ; i < PG_NUM ; i++ ) {
        va_i = va + (i << 12);
        vpn2 = (va_i << 25) >> 55;
        vpn1 = (va_i << 34) >> 55;
        vpn0 = (va_i << 43) >> 55;

        if(pgtbl[vpn2] % 2 == 0) {
            uint64 tmp = kalloc() - PA2VA_OFFSET;
            pgtbl[vpn2] = ((tmp >> 12) << 10) + 1;
        }
        uint64 *pgt1 = (uint64 *)((pgtbl[vpn2] >> 10) << 12);

        if (pgt1[vpn1] % 2 == 0) {

            uint64 tmp = kalloc() - PA2VA_OFFSET;
            pgt1[vpn1] = ((tmp >> 12) << 10) + 1;
        }
        uint64 *pgt0 = (uint64 *)(((pgt1[vpn1] >> 10) << 12)+PA2VA_OFFSET);

        pgt0[vpn0] = (((pa >> 12) + i) << 10) + (perm);
    }
}
```

4. 修改`mm.c`:

由于调用`mm_init`之前初始化虚拟内存，并且希望后续管理的内存都是从虚拟地址的基础分配，初始化的函数接收的起始结束地址需要调整为虚拟地址。

```c
void mm_init(void) {
    // kfreerange(_ekernel, (char *)PHY_END);
    kfreerange(_ekernel , (char *)VM_START + 128*1024*1024);
    printk("...mm_init done!\n");
}
```

实验结果

`make run` **观察运行结果**

```
kernel is running!
switch to [PID = 3 PRIORITY = 6 COUNTER = 1]
[PID = 3] is running! thread space begin at 0xffffffe007fbc000
switch to [PID = 5 PRIORITY = 2 COUNTER = 1]
[PID = 5] is running! thread space begin at 0xffffffe007fba000
switch to [PID = 21 PRIORITY = 9 COUNTER = 1]
[PID = 21] is running! thread space begin at 0xffffffe007faa000
switch to [PID = 23 PRIORITY = 3 COUNTER = 1]
[PID = 23] is running! thread space begin at 0xffffffe007fa8000
switch to [PID = 29 PRIORITY = 6 COUNTER = 1]
[PID = 29] is running! thread space begin at 0xffffffe007fa2000
switch to [PID = 14 PRIORITY = 6 COUNTER = 2]
[PID = 14] is running! thread space begin at 0xffffffe007fb1000
[PID = 14] is running! thread space begin at 0xffffffe007fb1000
switch to [PID = 26 PRIORITY = 2 COUNTER = 2]
[PID = 26] is running! thread space begin at 0xffffffe007fa5000
[PID = 26] is running! thread space begin at 0xffffffe007fa5000
switch to [PID = 6 PRIORITY = 3 COUNTER = 3]
[PID = 6] is running! thread space begin at 0xffffffe007fb9000
[PID = 6] is running! thread space begin at 0xffffffe007fb9000
[PID = 6] is running! thread space begin at 0xffffffe007fb9000
switch to [PID = 11 PRIORITY = 9 COUNTER = 3]
[PID = 11] is running! thread space begin at 0xffffffe007fb4000
[PID = 11] is running! thread space begin at 0xffffffe007fb4000
[PID = 11] is running! thread space begin at 0xffffffe007fb4000
switch to [PID = 18 PRIORITY = 3 COUNTER = 3]
[PID = 18] is running! thread space begin at 0xffffffe007fad000
[PID = 18] is running! thread space begin at 0xffffffe007fad000
[PID = 18] is running! thread space begin at 0xffffffe007fad000
switch to [PID = 19 PRIORITY = 8 COUNTER = 3]
[PID = 19] is running! thread space begin at 0xffffffe007fac000
[PID = 19] is running! thread space begin at 0xffffffe007fac000
[PID = 19] is running! thread space begin at 0xffffffe007fac000
switch to [PID = 22 PRIORITY = 7 COUNTER = 3]
[PID = 22] is running! thread space begin at 0xffffffe007fa9000
[PID = 22] is running! thread space begin at 0xffffffe007fa9000
[PID = 22] is running! thread space begin at 0xffffffe007fa9000
switch to [PID = 9 PRIORITY = 8 COUNTER = 4]
[PID = 9] is running! thread space begin at 0xffffffe007fb6000
[PID = 9] is running! thread space begin at 0xffffffe007fb6000
[PID = 9] is running! thread space begin at 0xffffffe007fb6000
[PID = 9] is running! thread space begin at 0xffffffe007fb6000
switch to [PID = 28 PRIORITY = 4 COUNTER = 4]
[PID = 28] is running! thread space begin at 0xffffffe007fa3000
```

```
[PID = 28] is running! thread space begin at 0xffffffe007fa3000
[PID = 28] is running! thread space begin at 0xffffffe007fa3000
[PID = 28] is running! thread space begin at 0xffffffe007fa3000
switch to [PID = 1 PRIORITY = 6 COUNTER = 5]
[PID = 1] is running! thread space begin at 0xffffffe007fbe000
[PID = 1] is running! thread space begin at 0xffffffe007fbe000
[PID = 1] is running! thread space begin at 0xffffffe007fbe000
[PID = 1] is running! thread space begin at 0xffffffe007fbe000
[PID = 1] is running! thread space begin at 0xffffffe007fbe000
switch to [PID = 8 PRIORITY = 1 COUNTER = 5]
[PID = 8] is running! thread space begin at 0xffffffe007fb7000
[PID = 8] is running! thread space begin at 0xffffffe007fb7000
[PID = 8] is running! thread space begin at 0xffffffe007fb7000
[PID = 8] is running! thread space begin at 0xffffffe007fb7000
[PID = 8] is running! thread space begin at 0xffffffe007fb7000
```

3 思考题

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

   ```
   wanghui@vmware:~/lab4$ cat System.map
   ffffffe000200000 A BASE_ADDR
   ffffffe000200584 T PrintTask
   ffffffe000203000 D TIMECLOCK
   ffffffe0002001c8 T __dummy
   ffffffe0002001d8 T __switch_to
   ffffffe000208fa0 B _ebss
   ffffffe000203008 D _edata
   ffffffe000208fa0 B _ekernel
   ffffffe00020216c R _erodata #
   ffffffe0002014b8 T _etext #
   ffffffe000204000 B _sbss
   ffffffe000203000 D _sdata
   ffffffe000200000 T _skernel
   ffffffe000202000 R _srodata #
   ffffffe000200000 T _start
   ffffffe000200000 T _stext #
   ......
   ```

2. 为什么我们在 `setup_vm` 中需要做等值映射?

   未分页时，访问的地址都是物理地址，分页后，监管者模式下访问的地址就是虚拟地址了，如果没有等值映射，会导致驱动开始运行时访问的物理地址不再是合理的物理地址，程序也就无法继续运行下去。

4 心得体会

- 按照实验指导，明确每一步的目的后，代码实现并不复杂。
- 操作系统实验与计算机体系结构内容联系紧密，本次实验关注从物理内存到虚拟内存的映射过程，事实上这对我们理解操作系统的实现有很大的帮助，通过这次实验对内存管理有了很好的理解。