浙江大学

本科实验报告

RV64 虚拟内存

课程名称：	操作系统
姓　　名：	陈杰伟
学　　院：	地球科学学院
专　　业：	地理信息科学
学　　号：	3200101205
指导老师：	寿黎但

2022 年 12 月 1 日

浙江大学实验报告

专业：地理信息科学
姓名：陈杰伟
学号：3200101205
日期：2022 年 12 月 1 日
地点：玉泉曹光彪-西 503

课程名称：操作系统　指导老师：寿黎但　成绩：
实验名称：RV64 虚拟内存　实验类型：编程实验　同组学生姓名：无

一、 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。

二、 实验环境

Ubuntu 20.04

三、 实验步骤

1. 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）。OS run 起来的时候, 其本身就是一个线程 idle 线程, 第一步为 idle 设置 task_struct。并将 current, task[0] 都指向 idle。

setup_vm

```
void setup_vm(void)
{

    /* 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。
    */
    // Page Table Entry 的权限 V | R | W | X 位设置为 1

    //注意：所有符号都被定义在虚拟地址，所以在启动虚拟地址需要反计算到物理地址(unsigned long *)((uint64)early_pgtbl - PA2VA_OFFSET)
    PHY_EARLY_PGTBL[((uint64)PHY_START << 25) >> 55] = (((uint64)PHY_START >> 30) << 28) + 0b0000001111; //等值映射

```

```
    PHY_EARLY_PGTBL[((uint64)VM_START << 25) >> 55] = (((uint64)PHY_START >> 30) << 28) +
        0b0000001111; //至高映射

    return;
}
```

完成上述映射之后，通过 relocate 函数，完成对 satp 的设置，以及跳转到对应的虚拟地址。

relocate

```
    relocate:
    # set ra = ra + PA2VA_OFFSET
    # set sp = sp + PA2VA_OFFSET (If you have set the sp before)

    li t0, PA2VA_OFFSET
    add ra, ra, t0
    add sp, sp, t0

    # set satp with early_pgtbl

    li t0, 0x8000000000000000
    la t1, early_pgtbl
    li t2, PA2VA_OFFSET
    sub t1, t1, t2
    srl t1, t1, 12
    add t0, t0, t1

    csrw satp, t0

    # flush tlb
    sfence.vma zero, zero

    # flush icache
    fence.i

    ret
```

2. setup_vm_final 的实现

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，可能需要修改 mm.c 中的代码，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。

mm_init

```
    void mm_init(void) {
        kfreerange(_ekernel, (char *)VM_KERNEL_END);
        printk("\n...mm_init done!\n");
    }
```

采用三级页表映射。

setup_vm_final

```
1   void setup_vm_final(void)
2   {
3       memset(swapper_pg_dir, 0x0, PGSIZE);
4
5       // mapping kernel text X|-|R|V
6       create_mapping(swapper_pg_dir, (uint64)_stext, PHY_ADR((uint64)_stext), (uint64)
            _etext - (uint64)_stext, (uint64)0b0000001011);
7
8       // mapping kernel rodata -|-|R|V
9       create_mapping(swapper_pg_dir, (uint64)_srodata, PHY_ADR((uint64)_srodata), (uint64
            )_erodata - (uint64)_srodata, (uint64)0b0000000011);
10
11      // mapping other memory -|W|R|V
12      create_mapping(swapper_pg_dir, (uint64)_sdata, PHY_ADR((uint64)_sdata), (uint64)
            VM_KERNEL_END - (uint64)_sdata, (uint64)0b0000000111);
13
14      csr_write(satp, (uint64)(0x8000000000000000 + ((uint64)PHY_ADR(swapper_pg_dir) >>
            12)));
15
16      // flush TLB
17      asm volatile("sfence.vma zero, zero");
18
19      // flush icache
20      asm volatile("fence.i");
21
22      return;
23  }
24
25  /* 创建多级页表映射关系 */
26  void create_mapping(uint64 *pgtbl, uint64 va, uint64 pa, uint64 sz, uint64 perm)
27  {
28      /*
29      pgtbl 为根页表的基地址
30      va, pa 为需要映射的虚拟地址、物理地址
31      sz 为映射的大小
32      perm 为映射的读写权限
33
34      创建多级页表的时候使用 kalloc() 来获取一页作为页表目录
35      使用 V bit 来判断页表项是否存在
36      */
37      uint64 offset = 0;
38
39      // 0级页表4kb映射
40      for (; offset < sz; offset += PGSIZE)
41      {
42          uint64 va_offset = va + offset;
43
44          //获取vpn
45          uint64 vpn[3] = {
```

```
46              (va_offset << 43) >> 55,
47              (va_offset << 34) >> 55,
48              (va_offset << 25) >> 55};
49
50          uint64 *pg_dir_l1 ,*pg_dir_l0;
51
52          //1gb映射
53          if (pgtbl[vpn[2]] == 0)
54          {
55              pg_dir_l1 = (uint64 *)kalloc();
56              pgtbl[vpn[2]] = (((uint64)PHY_ADR(pg_dir_l1) >> 12) << 10) + NOT_LEAF_PERM;
57          }
58          else
59              pg_dir_l1 = (uint64 *)(VIS_ADR((pgtbl[vpn[2]] >> 10) << 12));
60
61          if (pg_dir_l1[vpn[1]] == 0)
62          {
63              pg_dir_l0 = (uint64 *)kalloc();
64              pg_dir_l1[vpn[1]] = (((uint64)PHY_ADR(pg_dir_l0) >> 12) << 10) +
                    NOT_LEAF_PERM;
65          }
66          else
67              pg_dir_l0 = (uint64 *)(VIS_ADR((pg_dir_l1[vpn[1]] >> 10) << 12));
68
69          pg_dir_l0[vpn[0]] = (uint64)((((pa + offset) >> 12) << 10) + perm);
70      }
71
72      return;
73  }
```

在 head.S 中适当的位置调用 setup_vm_final 。

head.S 调用

```
1  la sp, boot_stack_top #将栈指针寄存器指向栈顶地址
2
3  call mm_init
4  call setup_vm_final
5
6  call task_init
```

四、 运行结果

各个线程正确运行在其虚拟页上，如图一

五、 思考题

1. 验证.text, .rodata 段的属性是否成功设置，给出截图。
对.text 段，程序可运行在高地址的.text 段，则其可执行
在 dummy 中调用如下代码

测试代码

```
printk("%d", *((uint64 *)_stext)); // read text
*((uint64 *)_stext) = 1; // write text
printk("%d", *((uint64 *)_stext));
```

并在上述行和 _traps 处打断点，开启调试

图 2 显示可以打印出 _stext 内存存储值，证明可读

图 3 显示向 _stext 地址写入触发了中断，中断原因是 store page fault 证明该页不可写

同样的对于.rodata 段

测试代码

```
printk("%d", *((uint64 *)_srodata)); // read text
*((uint64 *)_srodata) = 1; // write text
printk("%d", *((uint64 *)_srodata));
```

图 4 显示可以打印出 _srodata 内存存储值，证明可读

图 5 显示向 _srodata 地址写入触发了中断，中断原因是 store page fault 证明该页不可写

2. 为什么我们在 setup_vm 中需要做等值映射?

因为在写入 satp 开启虚拟内存后我们还需要做一系列操作和跳转至虚拟内存，而此时 PC 寄存器仍然指在物理地址上所以要对物理地址做等值映射才能确保程序继续执行

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。

Linux 采用懒加载策略（lazy allocation）在读到虚拟内存产生 page fault 后，系统从一个结构 vm_area 中读取到虚拟内存定义的范围，如果范围显示该虚拟内存合法，则在此时做虚拟内存映射，否则抛出 page fault

```
[PID = 14] is running. Thread space begin at 0xffffffe007fb1000

switch to [PID = 11 COUNTER = 3]
[PID = 11] is running. Thread space begin at 0xffffffe007fb4000
[PID = 11] is running. Thread space begin at 0xffffffe007fb4000

switch to [PID = 29 COUNTER = 3]
[PID = 29] is running. Thread space begin at 0xffffffe007fa2000
[PID = 29] is running. Thread space begin at 0xffffffe007fa2000

switch to [PID = 15 COUNTER = 4]
[PID = 15] is running. Thread space begin at 0xffffffe007fb0000
[PID = 15] is running. Thread space begin at 0xffffffe007fb0000
[PID = 15] is running. Thread space begin at 0xffffffe007fb0000

switch to [PID = 21 COUNTER = 4]
[PID = 21] is running. Thread space begin at 0xffffffe007faa000
[PID = 21] is running. Thread space begin at 0xffffffe007faa000
[PID = 21] is running. Thread space begin at 0xffffffe007faa000

switch to [PID = 4 COUNTER = 5]
[PID = 4] is running. Thread space begin at 0xffffffe007fbb000
[PID = 4] is running. Thread space begin at 0xffffffe007fbb000
[PID = 4] is running. Thread space begin at 0xffffffe007fbb000
[PID = 4] is running. Thread space begin at 0xffffffe007fbb000

switch to [PID = 5 COUNTER = 5]
[PID = 5] is running. Thread space begin at 0xffffffe007fba000
[PID = 5] is running. Thread space begin at 0xffffffe007fba000
[PID = 5] is running. Thread space begin at 0xffffffe007fba000
[PID = 5] is running. Thread space begin at 0xffffffe007fba000

switch to [PID = 18 COUNTER = 5]
[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
[PID = 18] is running. Thread space begin at 0xffffffe007fad000

switch to [PID = 25 COUNTER = 5]
[PID = 25] is running. Thread space begin at 0xffffffe007fa6000
[PID = 25] is running. Thread space begin at 0xffffffe007fa6000
[PID = 25] is running. Thread space begin at 0xffffffe007fa6000
[PID = 25] is running. Thread space begin at 0xffffffe007fa6000
```

图 1: 虚拟内存结果

```
switch to [PID = 1 COUNTER = 2]

Breakpoint 3, _traps () at entry.S:11
11          addi sp, sp, -33*16
=> 0xffffffe0002000c0 <_traps+0>:       13 01 01 df     addi    sp,sp,-528
(gdb) c
Continuing.
[PID = 1] is running. Thread space begin at 0xffffffe007fbe000

Breakpoint 1, dummy () at proc.c:75
75                  printk("%d", *((uint64 *)_stext)); // read text
=> 0xffffffe000200768 <dummy+168>:      97 37 00 00     auipc   a5,0x3
   0xffffffe00020076c <dummy+172>:      83 b7 87 91     ld      a5,-1768(a5) # 0xffffffe000203080
(gdb) c
Continuing.
12951
Breakpoint 2, dummy () at proc.c:76
76                  *((uint64 *)_stext) = 1;           // write text
=> 0xffffffe000200784 <dummy+196>:      97 37 00 00     auipc   a5,0x3
   0xffffffe000200788 <dummy+200>:      83 b7 c7 8f     ld      a5,-1796(a5) # 0xffffffe000203080
(gdb)
```

图 2: _stext 可读

```
问题  终端  调试控制台  JUPYTER  SQL CONSOLE

[PID = 1] is running. Thread space begin at 0xffffffe007fbe000

Breakpoint 1, dummy () at proc.c:75
75                  printk("%d", *((uint64 *)_stext)); // read text
=> 0xffffffe000200768 <dummy+168>:      97 37 00 00     auipc   a5,0x3
   0xffffffe00020076c <dummy+172>:      83 b7 87 91     ld      a5,-1768(a5) # 0xffffffe000203080
(gdb) c
Continuing.
12951
Breakpoint 2, dummy () at proc.c:76
76                  *((uint64 *)_stext) = 1;           // write text
=> 0xffffffe000200784 <dummy+196>:      97 37 00 00     auipc   a5,0x3
   0xffffffe000200788 <dummy+200>:      83 b7 c7 8f     ld      a5,-1796(a5) # 0xffffffe000203080
(gdb) c
Continuing.

Breakpoint 3, _traps () at entry.S:11
11          addi sp, sp, -33*16
=> 0xffffffe0002000c0 <_traps+0>:       13 01 01 df     addi    sp,sp,-528
(gdb) i r scause
scause         0xf      15
(gdb)
```

图 3: _stext 不可写

```
Breakpoint 3, _traps () at entry.S:11
11          addi sp, sp, -33*16
=> 0xffffffe0002000c0 <_traps+0>:       13 01 01 df     addi    sp,sp,-528
(gdb) c
Continuing.

switch to [PID = 1 COUNTER = 2]

Breakpoint 3, _traps () at entry.S:11
11          addi sp, sp, -33*16
=> 0xffffffe0002000c0 <_traps+0>:       13 01 01 df     addi    sp,sp,-528
(gdb) c
Continuing.
[PID = 1] is running. Thread space begin at 0xffffffe007fbe000

Breakpoint 1, dummy () at proc.c:75
75                  printk("%d", *((uint64 *)_srodata)); // read text
=> 0xffffffe000200768 <dummy+168>:      97 37 00 00     auipc   a5,0x3
   0xffffffe00020076c <dummy+172>:      83 b7 07 8c     ld      a5,-1856(a5) # 0xffffffe000203028
(gdb) c
Continuing.
774778378
Breakpoint 2, dummy () at proc.c:76
76                  *((uint64 *)_srodata) = 1;           // write text
=> 0xffffffe000200784 <dummy+196>:      97 37 00 00     auipc   a5,0x3
   0xffffffe000200788 <dummy+200>:      83 b7 47 8a     ld      a5,-1884(a5) # 0xffffffe000203028
(gdb)
```

图 4: _srodata 可读

```
Breakpoint 3, _traps () at entry.S:11
11          addi sp, sp, -33*16
=> 0xffffffe0002000c0 <_traps+0>:       13 01 01 df     addi    sp,sp,-528
(gdb) c
Continuing.
[PID = 1] is running. Thread space begin at 0xffffffe007fbe000

Breakpoint 1, dummy () at proc.c:75
75                  printk("%d", *((uint64 *)_srodata)); // read text
=> 0xffffffe000200768 <dummy+168>:      97 37 00 00     auipc   a5,0x3
   0xffffffe00020076c <dummy+172>:      83 b7 07 8c     ld      a5,-1856(a5) # 0xffffffe000203028
(gdb) c
Continuing.
774778378
Breakpoint 2, dummy () at proc.c:76
76                  *((uint64 *)_srodata) = 1;           // write text
=> 0xffffffe000200784 <dummy+196>:      97 37 00 00     auipc   a5,0x3
   0xffffffe000200788 <dummy+200>:      83 b7 47 8a     ld      a5,-1884(a5) # 0xffffffe000203028
(gdb) c
Continuing.

Breakpoint 3, _traps () at entry.S:11
11          addi sp, sp, -33*16
=> 0xffffffe0002000c0 <_traps+0>:       13 01 01 df     addi    sp,sp,-528
(gdb) i r scause
scause         0xf      15
(gdb)
```

图 5: _srodata 不可写