Operating system

Lab 2

Author: 吴杭

Date: 2024/11/17

Autumn-Fall 2024-2025 Semester

Table of Contents

Chapter 1: 实验流程

1.1) setup_vm 的实现

首先每个页表自身占据一页的内容，所以我们将页表所在的地址初始化为 0x0。然后我们在页表里面建立两个映射，第一个是等值映射（把当前的物理地址映射到物理地址），第二个是将虚拟地址的部分映射到物理地址上。

建立等值映射的原因在思考题中会解释。

根据框架注释的提示，代码实现如下：

```
void setup_vm() {
    // clear up early_pgtbl
    memset(early_pgtbl, 0x0, PGSIZE);

    // record first mapping
    int index = (PHY_START >> 30) & 0x1ff;
    early_pgtbl[index] = ((PHY_START >> 12) << 10) | 0xf;

    // record second mapping
    index = (VM_START >> 30) & 0x1ff;
    early_pgtbl[index] = ((PHY_START >> 12) << 10) | 0xf;
}
```

建立映射后，mm_init 所使用的地址变为虚拟地址，需要改变 mm_init 中使用的地址范围。

```
void mm_init(void) {
    kfreerange(_ekernel, (char *)PHY_END + PA2VA_OFFSET);
    printk("...mm_init done!\n");
}
```

1.2) relocate

我们首先需要将 ra 和 sp 的值移动到后续将要读取的虚拟地址上。PA2VA_OFFSET 是一个比较大的值，所以这里我们的处理如下。

```
# load the value PA2VA_OFFSET in a reg
lui t0, 0xffdf8
slli t0, t0, 16

# set ra = ra + PA2VA_OFFSET
add ra, ra, t0

# set sp = sp + PA2VA_OFFSET
add sp, sp, t0
```

这里要完成对 satp 寄存器的写入操作，根据文档中的介绍，我们需要把 mode 设置为 8，对应于 Sv39 的模式，然后把 ASID 设置为 0，然后在 PPN 中写入页表的地址。

```
    # need a fence to ensure the new translations are in use
    sfence.vma zero, zero

    # set mode value
```

```
    li t1, 0x8
    slli t1, t1, 60

    # set asid value
    li t2, 0
    slli t2, t2, 44

    # set PPN
    la t3, early_pgtbl
    srli t3, t3, 12

    # merge these three values
    or t3, t3, t2
    or t3, t3, t1

    # set satp
    csrw satp, t3

    ret
```

1.3) setup_vm_final 的实现

为方便后续程序的编写，预先定义 setup_vm_final 中将使用的常量。

```
#define PRIV_V (1 << 0)
#define PRIV_R (1 << 1)
#define PRIV_W (1 << 2)
#define PRIV_X (1 << 3)
#define PRIV_U (1 << 4)
#define PRIV_G (1 << 5)
#define PRIV_A (1 << 6)
#define PRIV_D (1 << 7)

#define MODE_SV39 8
```

setup_vm_final 借助以下函数 create_mapping 创建映射关系。create_mapping 从需要映射的虚拟地址中取出三级页表的索引，根据索引获取对应的页表，最后将物理地址和权限写入页表项中。

```
void create_mapping(uint64_t *pgtbl, uint64_t va, uint64_t pa, uint64_t sz,
                    uint64_t 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，单位为字节
   * perm 为映射的权限（即页表项的低 8 位）
   *
   * 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
   **/
  for (int i = 0; i < sz; ++i, va += 0x1000, pa += 0x1000) {
    // virtual page number
    uint64_t vpn0 = (va >> 12) & 0x1ff;
```

```
    uint64_t vpn1 = (va >> 21) & 0x1ff;
    uint64_t vpn2 = (va >> 30) & 0x1ff;

    uint64_t *pgtbl1 = get_pgtable(pgtbl, vpn2);
    uint64_t *pgtbl0 = get_pgtable(pgtbl1, vpn1);

    // check if page already exists
    if (!(pgtbl0[vpn0] & PRIV_V)) {
      // write pa and perm
      pgtbl0[vpn0] = perm | ((pa >> 2) & 0x3ffffffffffc00);
    }
  }
}
```

其中，get_pgtable 函数用于从上级页表中获取下级页表的地址。如果对应的下级页表不存在，则新建一个页表，将其地址转换为物理地址后写入上级页表中。

```
uint64_t *get_pgtable(uint64_t *pgtbl, uint64_t vpn) {
  // check if page already exists
  if (pgtbl[vpn] & PRIV_V) { // exists
    return (uint64_t *)((pgtbl[vpn] & 0x3ffffffffffc00) << 2);
  } else { // does not exist
    uint64_t *new_pgtbl = kalloc();
    memset(new_pgtbl, 0x0, PGSIZE);
    uint64_t new_pgtbl_pa = (uint64_t)new_pgtbl - PA2VA_OFFSET;
    pgtbl[vpn] = ((uint64_t)new_pgtbl_pa >> 2) | PRIV_V;
    return new_pgtbl;
  }
}
```

setup_vm_final 借助 create_mapping 函数将内核的 text 段、rodata 段与 data 段映射到内核的虚拟地址空间中。完成多级页表的建立后，计算得到页表的物理地址，并将其写入 satp 中（仍为 Sv39 模式），最后刷新 TLB。

```
/* swapper_pg_dir: kernel pagetable 根目录, 在 setup_vm_final 进行映射 */
uint64_t swapper_pg_dir[512] __attribute__((__aligned__(0x1000)));

extern uint64_t _stext, _srodata, _sdata;

void setup_vm_final() {
  memset(swapper_pg_dir, 0x0, PGSIZE);

  // No OpenSBI mapping required

  // mapping kernel text X|-|R|V
  uint64_t size_text = ((uint64_t)&_srodata - (uint64_t)&_stext) >> 12;
  create_mapping(swapper_pg_dir, (uint64_t)&_stext,
                 (uint64_t)&_stext - PA2VA_OFFSET, size_text,
                 PRIV_X | PRIV_R | PRIV_V);

  // mapping kernel rodata -|-|R|V
  uint64_t size_rodata = ((uint64_t)&_sdata - (uint64_t)&_srodata) >> 12;
  create_mapping(swapper_pg_dir, (uint64_t)&_srodata,
```

```
                (uint64_t)&_srodata - PA2VA_OFFSET, size_rodata,
                PRIV_R | PRIV_V);

  // mapping other memory -|W|R|V
  create_mapping(swapper_pg_dir, (uint64_t)&_sdata,
                (uint64_t)&_sdata - PA2VA_OFFSET,
                32768 - size_text - size_rodata, PRIV_W | PRIV_R | PRIV_V);

  // set satp with swapper_pg_dir

  // physical address of swapper_pg
  uint64_t swapper_pg_dir_pa = (uint64_t)swapper_pg_dir - PA2VA_OFFSET;
  uint64_t satp =
      ((uint64_t)MODE_SV39 << 60) | ((uint64_t)swapper_pg_dir_pa >> 12);
  asm volatile("csrw satp, %0" ::"r"(satp));

  // flush TLB
  asm volatile("sfence.vma zero, zero");
  return;
}
```

在 `setup_vm_final` 中，我们需要申请页面以建立多级页表，因此在调用前需要先通过 `mm_init` 将内存管理初始化。

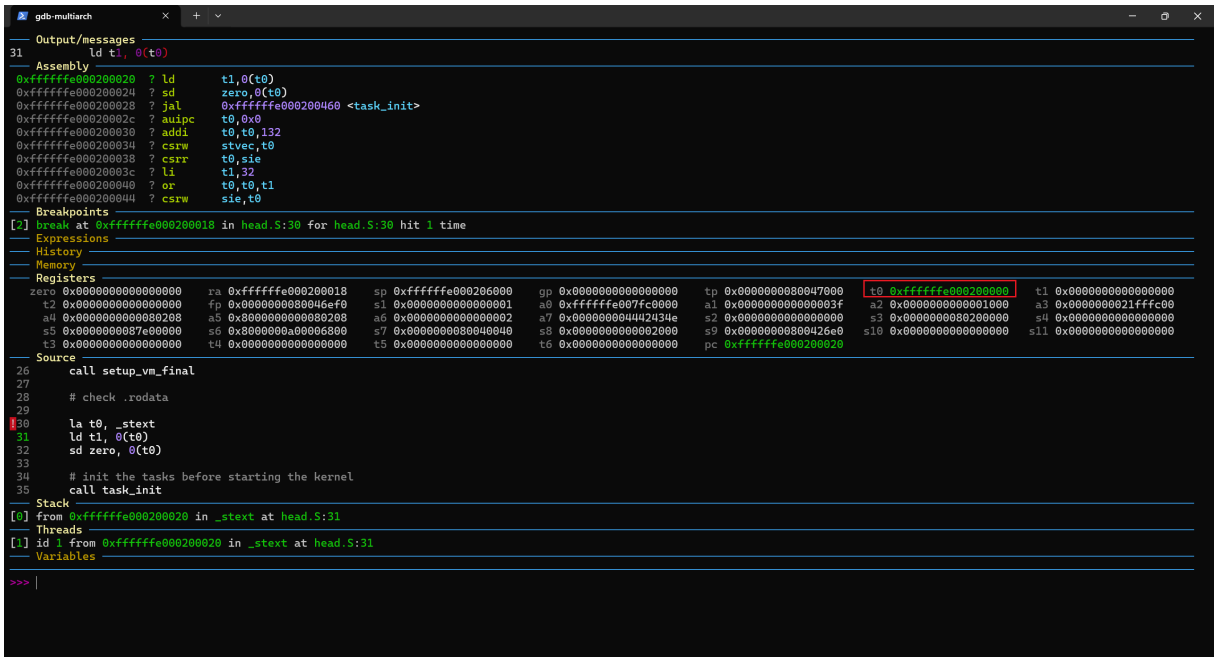
```
_start:
    # -----------------
    # - your code here -
    la sp, boot_stack_top

    # call setup_vm
    call setup_vm

    # call relocate
    call relocate

    # initialize the memory management
    call mm_init

    call setup_vm_final

    # ...
```

Chapter 2: 思考题

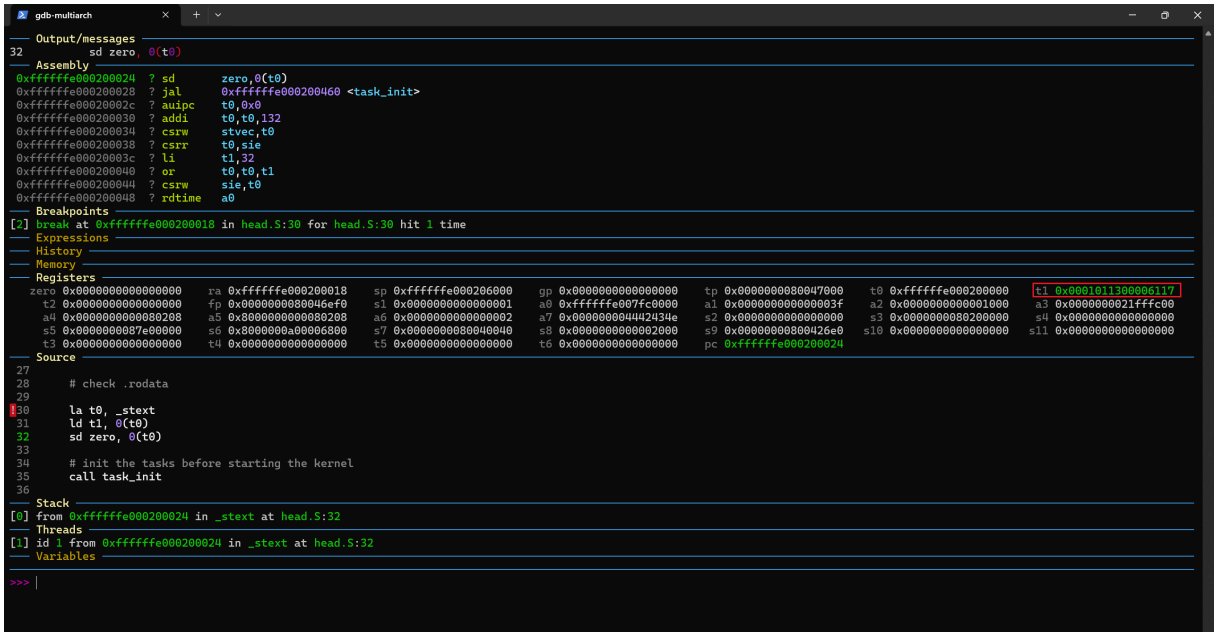
2.1) 1.

2.1.1) `.text` 的测试

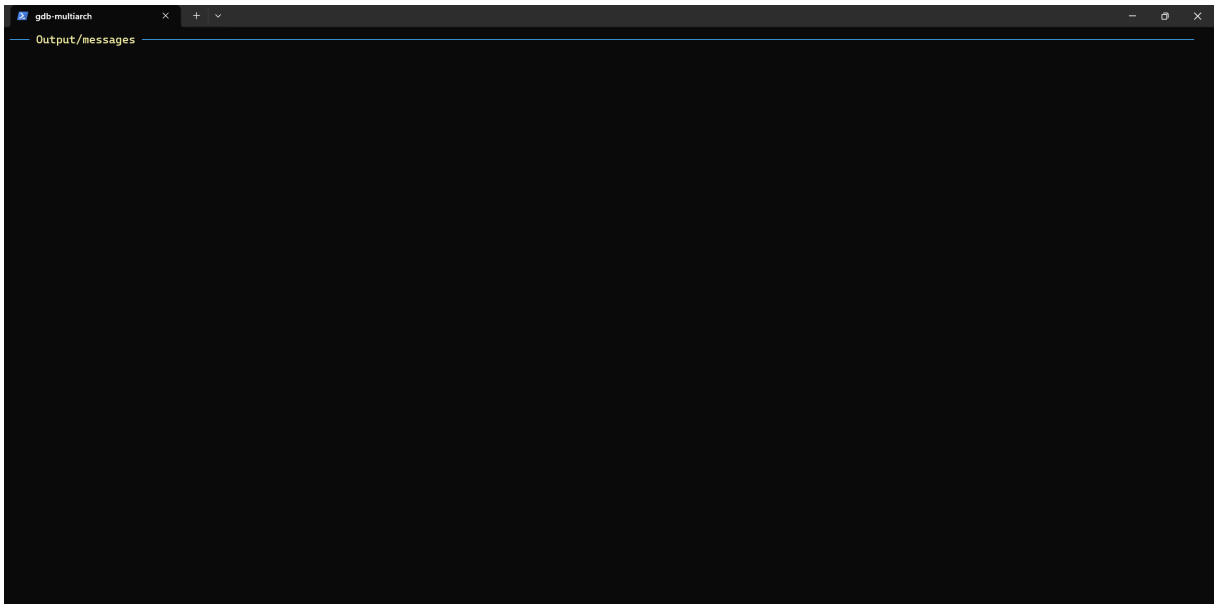
可以看到 `la t0, _stext` 成功读入了`_stext` 对应的地址



可以看到 `ld t1, 0(t0)`成功读入了`_stext` 处的数据，说明 R 属性被正确设置了。

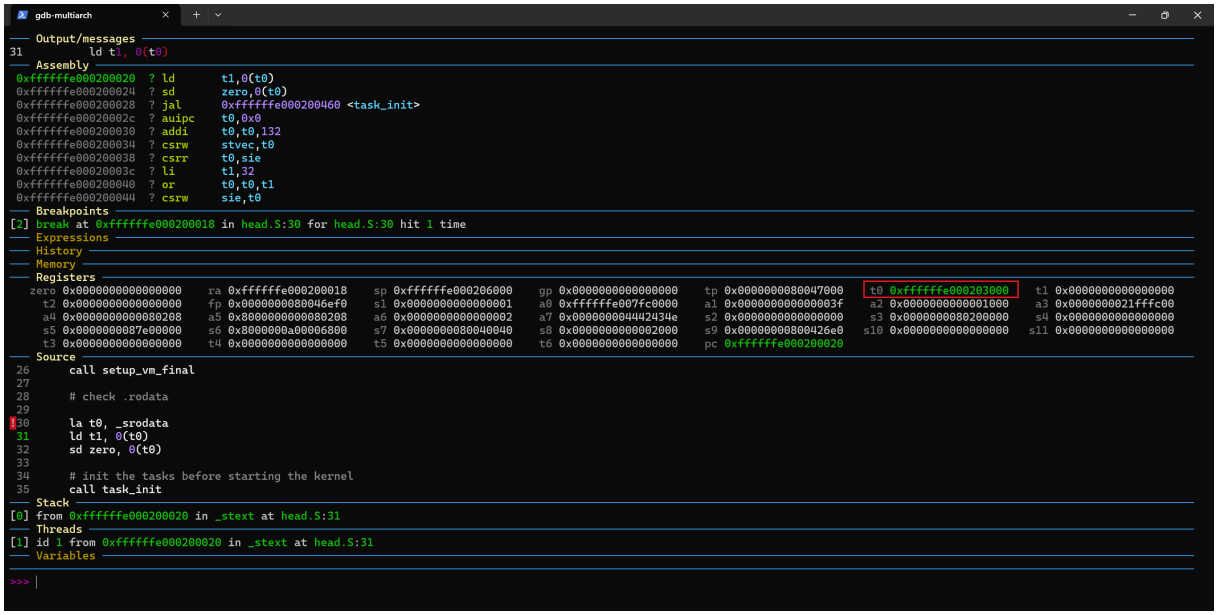


当我们要执行 `sd zero, 0(t0)`的时候发现，程序崩溃了，说明 W 属性确实没有被允许。

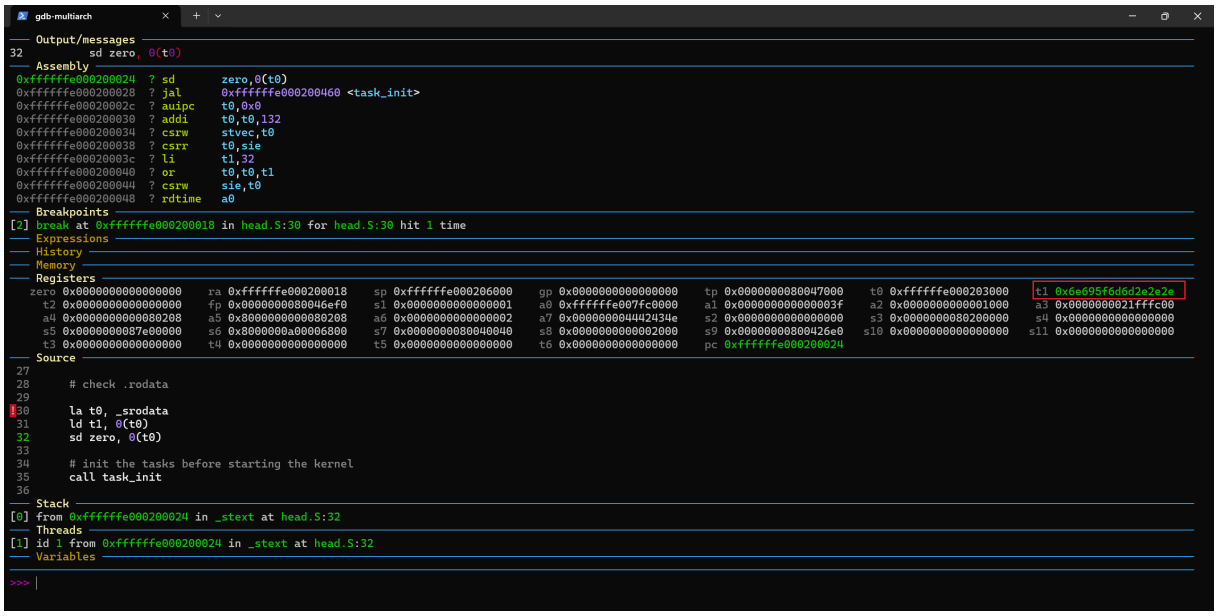


2.1.2) `.rodata` 的测试

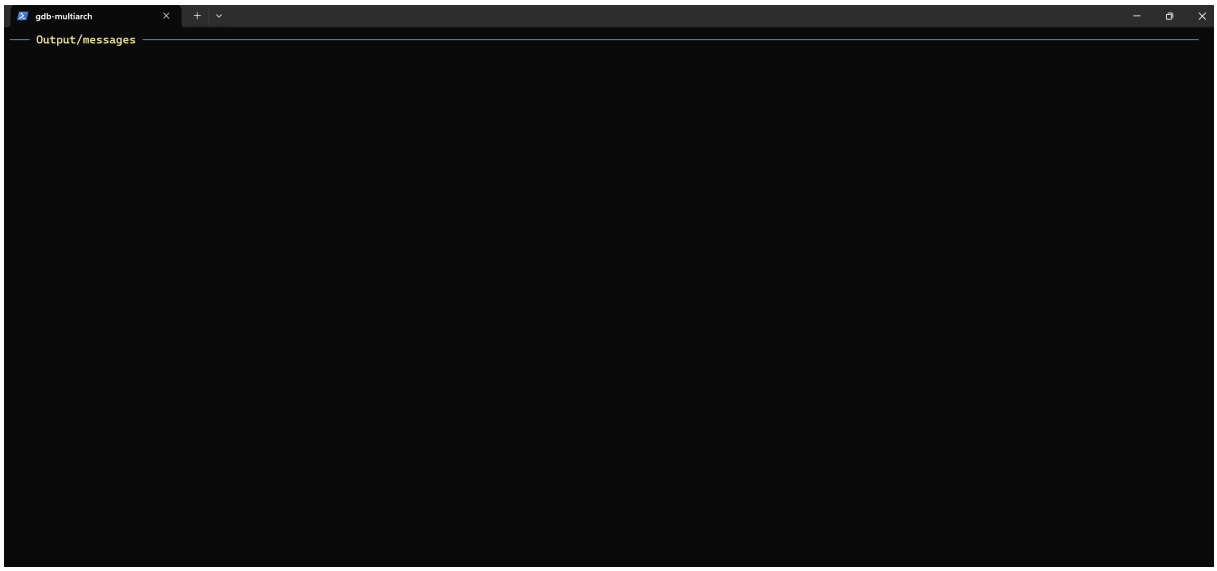
可以看到 `la t0, _srodata` 成功读入了`_srodata` 对应的地址



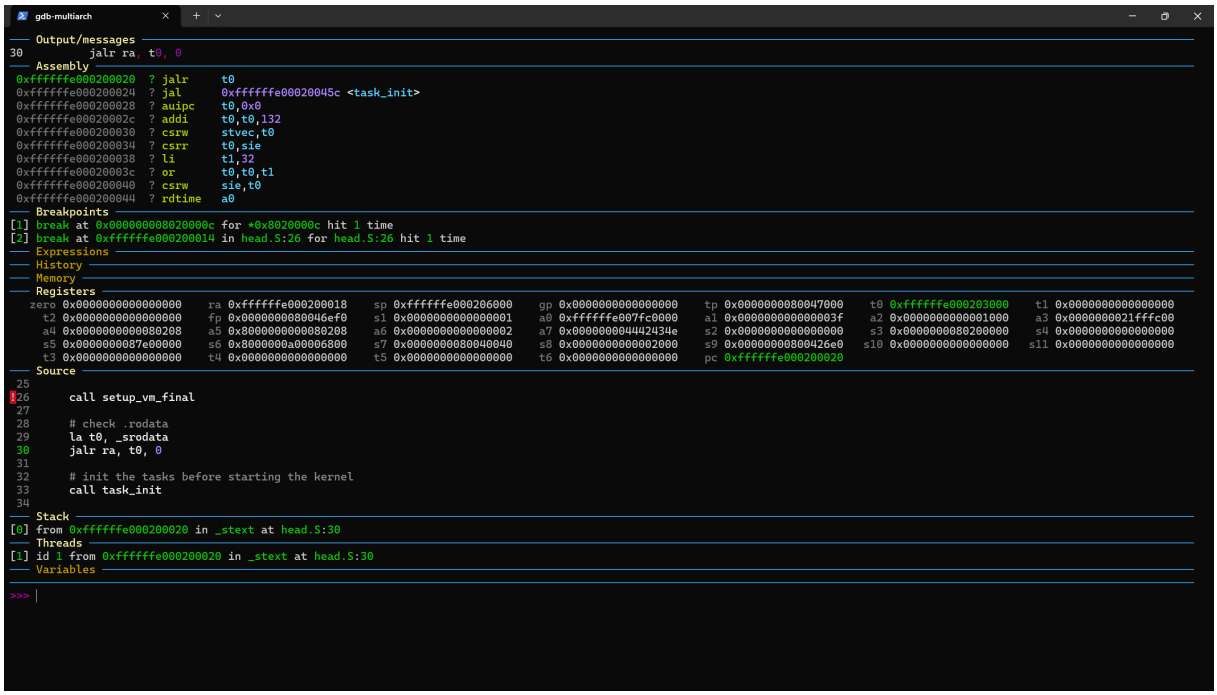
可以看到 `ld t1, 0(t0)`成功读入了`_srodata` 处的数据，说明 R 属性被正确设置了。

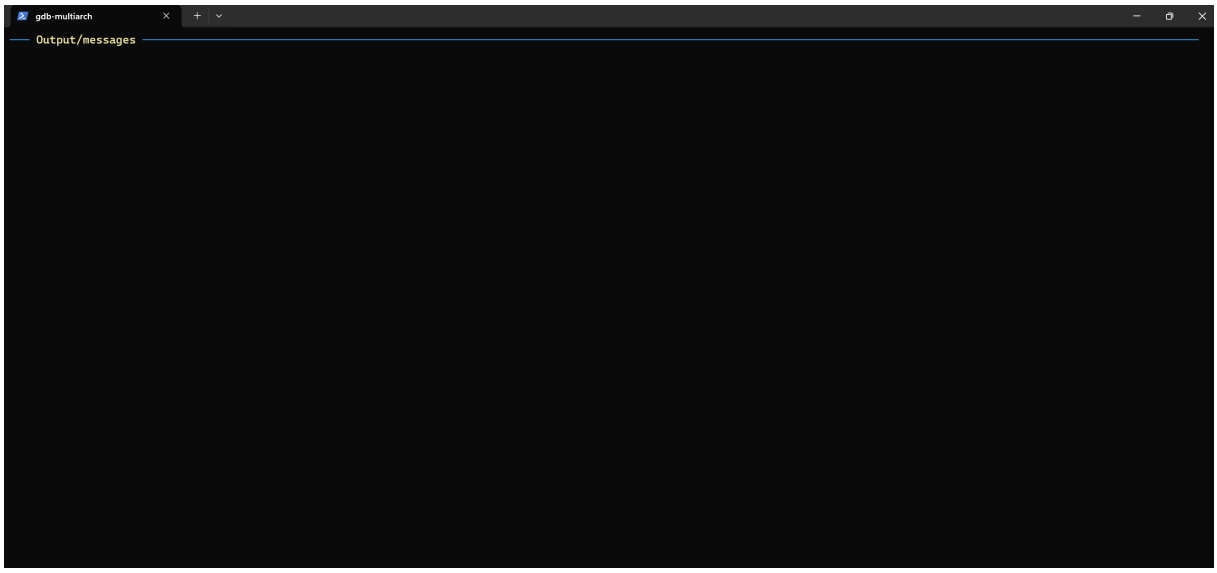


当我们要执行 `sd zero, 0(t0)`的时候发现，程序崩溃了，说明 W 属性确实没有被允许（置为 0）。



当我们要执行 `jalr ra, t0, 0` 的时候发现，程序崩溃了，说明 X 属性确实没有被允许（置为 0）。





2.2) 2.

2.2.1) a.

本次实验中建立等值映射的原因在于，在我们设置 `satp` 之后，我们的 PC 仍然在物理地址上，程序此时认为自己处于“虚拟地址”上，就会尝试把当前的地址通过页表查找到“物理地址”，如果不建立等值映射，程序就找不到对应的映射后的地址了。

2.2.2) b.

linux 的内核启动部分在设置 `satp` 附近的逻辑如下：

1. 首先设置 `sie` 和 `sip`
2. load 全局指针到 `gp` 中
3. 通过禁用 `FPU` 来防止

4. 清空 bss 段
5. 执行 setup_vm 来初始化页表
6. 进入 relocate 的代码
7. 先把 ra 中的物理地址改为虚拟地址，这是为了在改变寻址方式之后，返回时能到达正确的地址
8. 将 stvec 修改为设置 satp 后的第一条指令的虚拟地址
9. 计算好 swapper_pg_dir 的值，但是先不写入到 satp 中。
10. 将 trampoline_pg_dir 的值对应的物理页号以及 mode 一起写入到 satp 中，然后此时寻址方式就变成根据虚拟地址来寻址了，但是此时 PC 仍然在物理地址上，由于没有建立等值映射，所以会触发一个 page fault。然后程序会跳转到 stvec 中记录的地址执行代码，而此时该地址恰好是设置 satp 的下一条指令。
11. 重新设置 stvec
12. 重新把 swapper_pg_dir 和 mode 一起写入到 satp 中。
13. 跳转到 start_kernel

2.2.3) c.

linux 之所以可以不进行等值映射，是因为当我们找不到对应地址时，会触发 page fault 的中断，而处理中断的程序地址恰好被设置为了设置 satp 之后的下一条指令，所以能恰好继续执行代码。

2.2.4) d.

trampoline_pg_dir 相当于我们实验中实现的 early_pgtbl，他不是最终的页表。而 swapper_pg_dir 则是最终被用来转化虚拟地址到物理地址的页表。trampoline_pg_dir 是在第一次设置完 stvec 之后写入到 satp 中的。然后等到触发了 page fault 之后，swapper_pg_dir 才会被写入到 satp 中。

2.2.5) e.

只需要修改 relocate 处的代码如下即可

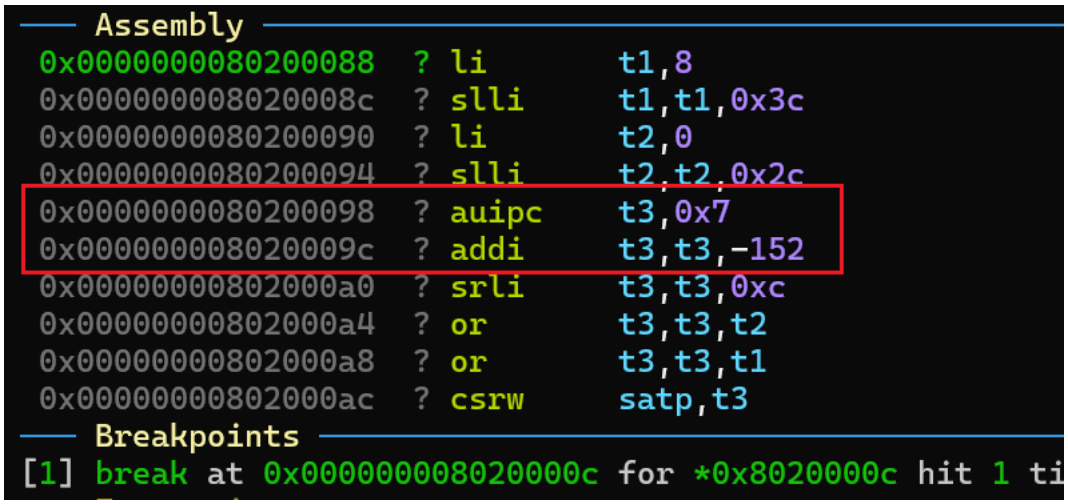
```
relocate:
    # set ra = ra + PA2VA_OFFSET
    # set sp = sp + PA2VA_OFFSET (If you have set the sp before)

    ######################
    #   YOUR CODE HERE   #
    ######################

    # load the value PA2VA_OFFSET in a reg
    lui t0, 0xffdf8
    slli t0, t0, 16

    # set ra = ra + PA2VA_OFFSET
    add ra, ra, t0

    # set sp = sp + PA2VA_OFFSET
```

```
    add sp, sp, t0

    csrr t1, stvec
    la t1, 1f
    add t1, t1, t0
    csrw stvec, t1

    # set satp with early_pgtbl

    # need a fence to ensure the new translations are in use
    sfence.vma zero, zero

    # set mode value
    li t1, 0x8
    slli t1, t1, 60

    # set asid value
    li t2, 0
    slli t2, t2, 44


    # set PPN
    la t3, early_pgtbl
    srli t3, t3, 12

    # merge these three values
    or t3, t3, t2
    or t3, t3, t1

    # set satp
    csrw satp, t3
.align 2
1:
    ret
```

Chpater 3: 心得体会

3.1) 遇到的问题

- 一开始不是很理解代码中的那些符号的地址是怎么对应上的，比如我在设置 `satp` 之前，`la t3, early_pgtbl` 的时候，得到的到底是 `early_pgtbl` 的物理地址还是虚拟地址? 经过 gdb 的调试和分析，我逐渐理解了，load 这个符号的地址实际上是根据和 PC 的相对距离来做的，这样如果 PC 在物理地址上，那么 load 出来的就是物理地址，如果 PC 在虚拟地址上，那么 load 出来的就是虚拟地址。本质在当前的 PC 和这个符号的相对位置，所以无论是通过虚拟地址还是物理地址寻址，都能找到正确的符号的地址。



- 一开始不明白为什么 setup_vm 里面需要先设置一个等值映射，后来通过和同学的讨论得知，是避免设置完 satp 之后 PC 在物理地址上找不到对应的物理地址的情况。

3.2) 心得体会

这次实验个人认为对于虚拟内存的概念的理解很有帮助，但是实验文档感觉思维略微有点跳跃，有些地方感觉没说清楚，比如为什么要先进行 `setup_vm` 然后再做 `setup_vm_final`？对于第一次做这个实验的同学可能会觉得一头雾水。再比如为什么要先做等值映射？在没有任何提示的情况下，也不知道接下来写入 `satp` 的过程的时候，看到这个东西更加是十分迷惑。我认为就算要把这个地方作为一个考点，也要在文档里给出相应提示，至少明确指出这个地方是需要同学们思考原因的（结果在最后思考题的部分才指出，这样让读文档的体验感非常不好）。因为这种在第一次看起来比较反常的操作，会影响部分做实验的同学怀疑是自己的理论知识哪里还没看完整，然后又花大量时间去查阅文档，结果一无所获，非常影响实验的体验感。

Declaration

We hereby declare that all the work done in this lab 3 is of our independent effort.