浙江大学

本科实验报告

课程名称：	操作系统
姓　　名：	董佳鑫
学　　院：	计算机学院
系：	计算机系
专　　业：	计算机科学与技术
学　　号：	3210102181
指导教师：	寿黎但

Lab 4：RV64 用户态程序

实验目的

1.创建用户态进程，并设置 sstatus 来完成内核态转换至用户态。
2.正确设置用户进程的用户态栈和内核态栈， 并在异常处理时正确切换。
3.补充异常处理逻辑，完成指定的系统调用（SYS_WRITE, SYS_GETPID）功能。

实验过程和操作步骤

1. 创建用户态进程

首先同步代码，进行一些基本的修改。创建好实验环境后，首先修改 task_init。其他变量赋值和之前相同，对于新的变量，按照实验要求进行赋值。

```
uint64_t mode = 8;
mode = mode << 60;
task[i]->pgd = mode | ((unsigned long)pgtbl-PA2VA_OFFSET) >> 12;
task[i]->thread.sepc = USER_START;
task[i]->thread.sstatus = (csr_read(sstatus) | 0x40020) & 0xfffffffffffffeff;
task[i]->thread.sscratch = USER_END;
```

将 sepc 变量赋值为 USER_START，即用户态初始虚拟地址；sscratch 赋值为 USER_END,即用户态栈地址；sstatus 中要将 SPP 设为 0，其余两个 flag 设为 1；由于 task->pgd 是用于后续切换页表时修改 satp 寄存器的值，因此这里直接将 task->pgd 的值改成 stap 的格式并保存为当前页表的 PPN。

```
pagetable_t pgtbl = (pagetable_t)kalloc();
memcpy((void *)(pgtbl), (void *)(&swapper_pg_dir), PGSIZE);
uint64_t va,pa;
va = USER_START;
pa = (unsigned long)(_sramdisk)-PA2VA_OFFSET;
create_mapping_sub(pgtbl, va, pa, (unsigned long)(_eramdisk)-(unsigned long)(_sramdisk), 31);
va = USER_END - PGSIZE;
pa = (uint64_t)alloc_page() - PA2VA_OFFSET;
create_mapping_sub(pgtbl,va,pa,PGSIZE,23);
```

由于每个用户态进程都需要自己的页表，因此需要分别创建页表并建立映射。首先为页表申请一块内存，并把内核页表复制过去。然后将 uapp 所在的页面映射到起始虚拟地址 USER_START。注意 uapp 的起始地址和终止地址分别为_sramdisk 和_eramdisk。之后设置用户态栈，将 USER_END 设置为栈指针，再申请一块内存作为栈空间，同样建立一次映射。

完成 task_init 的修改后，下面修改__switch_to 函数，由于 struct task 新增了四个变量（sepc sstatus sscratch 和 satp），因此需要对这四个变量也进行上下文切换。并且要在切换完页表后刷新 TLB 和 icache。

```
csrr t5, sepc
sd t5, 112(a3)
csrr t5, sstatus
sd t5, 120(a3)
csrr t5, sscratch
sd t5, 128(a3)
csrr t5, satp
sd t5, 136(a3)
```

```
ld t5, 112(a4)
csrw sepc, t5
ld t5, 120(a4)
csrw sstatus, t5
ld t5, 128(a4)
csrw sscratch, t5
ld t5, 136(a4)
csrw satp, t5


sfence.vma zero, zero

# flush icache
fence.i
```

2.修改中断入口/返回逻辑_trap 以及中断处理函数 trap_handler

首先修改__dummy 函数，将 sscratch 和 sp 变量的值交换即可。便完成了 S 模式转换到 U 模式。

然后修改_trap 函数,这里的处理和__dummy 类似，但是需要判断 sscratch 是否为 0，如果为 0 则不进行变量值交换。

```
    csrr t0,sscratch
    beq t0,x0,_traps_switch
    csrw sscratch,sp
    mv sp,t0
_traps_switch:
        # 1. save 32 registe
    addi sp, sp, -248
    sd x1, 0(sp)
```

```
    csrr t0,sscratch
    beq t0,x0,_traps_end
    csrw sscratch,sp
    mv sp,t0
_traps_end:
    sret
```

下面我们要增加新的中断处理，修改 trap_handler 函数，增加新的参数 struct pt_regs*（因此在 entry.s 中，在 call trap_handler 之前要给 a2 赋值为 sp）。这个结构体可以取到寄存器的值，内部变量应该包括全部的常规寄存器和 sepc sstatus 寄存器。然后在 trap_handler 中增加 ECALL_FROM_U_MODE exception 处理，通过查阅手册可知，这种类型的中断对应的 scause = 8，因此添加对应的中断处理即可。

3.添加系统调用

下面具体实现 ECALL_FROM_U_MODE 的中断处理逻辑。根据手册要求完成即可。

```c
void syscall(struct pt_regs* regs){

    if(regs->reg[17 - 1] == SYS_WRITE){
        uint64_t n = regs->reg[12 - 1];
        if(regs->reg[10 - 1] == 1){
         char *buff = (char *)regs->reg[11 - 1];
         uint64_t i;
         for(i = 0;i < n;i++)
           printk("%c",buff[i]);
         regs->reg[10 - 1] = n;
        } else regs->reg[10 - 1] = 0;

    }else if(regs->reg[17 - 1] == SYS_GETPID){
        regs->reg[10 - 1] = current->pid;
    }

}
```

这里 regs->reg[17]对应 x17，即 a7，用于传入系统调用号；regs->reg[10]对应 x10，即 a0，用于作为返回值。然后在 trap_handler 中 scause = 8 的情况下调用该 syscall 函数即可。

4. 修改 head.s 和 start_kernel 并测试二进制文件

按照要求修改 head.s 和 start_kernel，就初步完成了该实验，我们可以进行测试。

```
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
...setup_vm done!
...buddy_init done!
...setup_vm_final done!
...proc_init done!
[S-MODE] 2022 Hello RISC-V
[U-MODE] pid: 1, sp is 0000003fffffffe0, this is print No.1
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[U-MODE] pid: 4, sp is 0000003fffffffe0, this is print No.2
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[U-MODE] pid: 3, sp is 0000003fffffffe0, this is print No.3
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[U-MODE] pid: 3, sp is 0000003fffffffe0, this is print No.4
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
```

5. 添加 ELF 支持

接下来我们要将二进制文件更换为 ELF 文件，并做一定的修改，继续输出正确结果。首先将 upp.s 中的 payload 修改为 ELF 文件 uapp，然后阅读手册并查阅资料完成 load_program 函数。

```c
for (int i = 0; i < phdr_cnt; i++) {
    phdr = (Elf64_Phdr*)(phdr_start + sizeof(Elf64_Phdr) * i);
    if (phdr->p_type == PT_LOAD) {
        // alloc space and copy content
        // do mapping
        uint64_t pgnum = (PGOFFSET(phdr->p_vaddr) + phdr->p_memsz + PGSIZE - 1)/PGSIZE;
        uint64_t pages = alloc_pages(pgnum);
        uint64_t start_addr = (uint64_t)(_sramdisk + phdr->p_offset);
        memcpy((void *)(pages + phdr->p_vaddr - PGROUNDDOWN(phdr->p_vaddr)),(void *)start_addr,phdr->p_memsz);
        create_mapping_sub((uint64_t *)pgtbl,(phdr->p_vaddr),(pages - PA2VA_OFFSET),pgnum*PGSIZE,31);
    }
}
```

```c
uint64_t va,pa;
va = USER_END - PGSIZE;
pa = (uint64_t)alloc_page() - PA2VA_OFFSET;
create_mapping_sub(pgtbl,va,pa,PGSIZE,23);
uint64_t mode = 8;
mode = mode << 60;
task->pgd = mode | ((unsigned long)pgtbl-PA2VA_OFFSET) >> 12;
// pc for the user program
task->thread.sepc = ehdr->e_entry;
// sstatus bits set
task->thread.sstatus = (csr_read(sstatus) | 0x40020) & 0xffffffffffffeff;
// user stack for user program
task->thread.sscratch = USER_END;
```

该函数需要将 LOAD 段加载到内存中，可以借助 memcpy 函数进行加载，然后重新修改映射函数参数，设置为正确的起始终止地址。此外 sepc 也要做一定的修改。

完成以后，再次编译测试，得到如下的结果，测试通过。

```
...setup_vm done!
...buddy_init done!
...setup_vm_final done!
...proc_init done!
[S-MODE] 2022 Hello RISC-V
[U-MODE] pid: 1, sp is 0000003fffffffe0, this is print No.1
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[U-MODE] pid: 4, sp is 0000003fffffffe0, this is print No.1
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[U-MODE] pid: 4, sp is 0000003fffffffe0, this is print No.2
[U-MODE] pid: 3, sp is 0000003fffffffe0, this is print No.1
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[U-MODE] pid: 3, sp is 0000003fffffffe0, this is print No.2
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
[U-MODE] pid: 2, sp is 0000003fffffffe0, this is print No.1
[S-MODE] Supervisor Mode Timer Interrupt
[S-MODE] Supervisor Mode Timer Interrupt
```

思考题

1. 我们在实验中使用的用户态线程和内核态线程的对应关系是怎样的？（一对一，一对多，多对一还是多对多）

   一对一。每一个用户线程都创建了一个内核线程对应，然后载入 ELF 文件。

2. 为什么 Phdr 中，p_filesz 和 p_memsz 是不一样大的？

   p_filesz 是 segment 在文件中占的大小，而 p_memsz 是在内存中占的大小，

主要有区别的地方在于未初始化数据在文件中不占用大小而在内存中会占用大小，因此 p_memsz≥p_filesz。

3.为什么多个进程的栈虚拟地址可以是相同的？用户有没有常规的方法知道自己栈所在的物理地址？

这就是虚拟内存的意义，让不同进程独立运行，仿佛感受不到其他进程。这主要是因为每个进程都有自己的页表，映射关系也不同，尽管栈虚拟地址相同，但他们各种经过页表映射后会指向不同的物理地址。

一般情况下，用户访问到的都是虚拟地址，物理地址往往是不可见的。并且页表一般不会暴露给用户，因此难以得到栈的物理地址。

讨论心得

本次实验与上一次实验比较密切，都是需要深刻理解各种地址的含义，尤其是要区分虚拟地址和物理地址。通过完成本次实验，我再一次深刻体会到了虚拟内存的重要意义，将不同的进程进行有效的隔离。本次实验难度依然较大，需要处理的细节很多，其中印象较为深刻的困难是，对于 uapp 和用户态栈建立映射，起初我使用上个实验的 create_mapping 函数却总是卡死，经过大量调试和思考后发现，上一个实验的三级映射没有建立等值映射，因此再次使用 create_mapping 建立映射时需要进行修改，将 pte 中的值增加 PA2VA_OFFSET，这样就能正确建立映射。此外在对 ELF 建立映射时要注意对齐问题，即映射起点要对齐 4kB。总之这次实验理解比较困难，实现细节要求也比较高，令人记忆深刻，收获颇丰。