

**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。

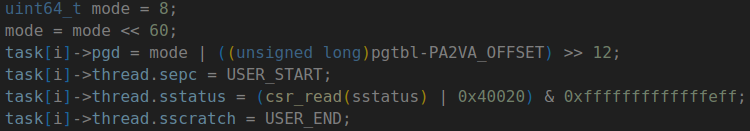
2.正确设置用户进程的用户态栈和内核态栈， 并在异常处理时正确切换。

3.补充异常处理逻辑，完成指定的系统调用（SYS\_WRITE, SYS\_GETPID）功能。

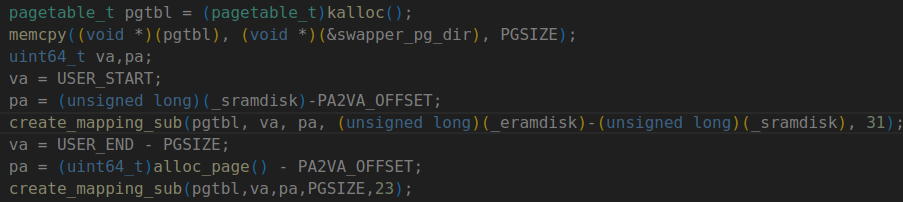
实验过程和操作步骤

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。

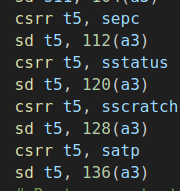


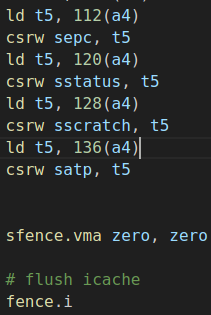
将sepc变量赋值为USER\_START，即用户态初始虚拟地址；sscratch赋值为USER\_END,即用户态栈地址；sstatus中要将SPP设为0，其余两个flag设为1；由于task->pgd是用于后续切换页表时修改satp寄存器的值，因此这里直接将task->pgd的值改成stap的格式并保存为当前页表的PPN。



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。

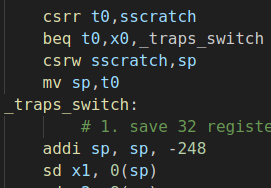


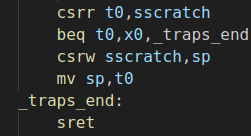


2.修改中断入口/返回逻辑\_trap以及中断处理函数trap\_handler

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。

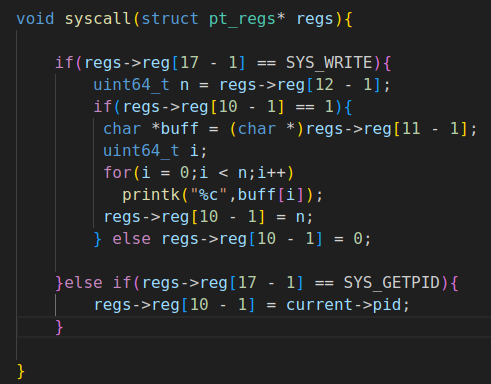




下面我们要增加新的中断处理，修改trap\_handler函数，增加新的参数struct pt\_regs\*（因此在entry.s中，在call trap\_handler之前要给a2赋值为sp）。这个结构体可以取到寄存器的值，内部变量应该包括全部的常规寄存器和sepc sstatus 寄存器。然后在trap\_handler中增加ECALL\_FROM\_U\_MODE exception处理，通过查阅手册可知，这种类型的中断对应的scause = 8，因此添加对应的中断处理即可。

3.添加系统调用

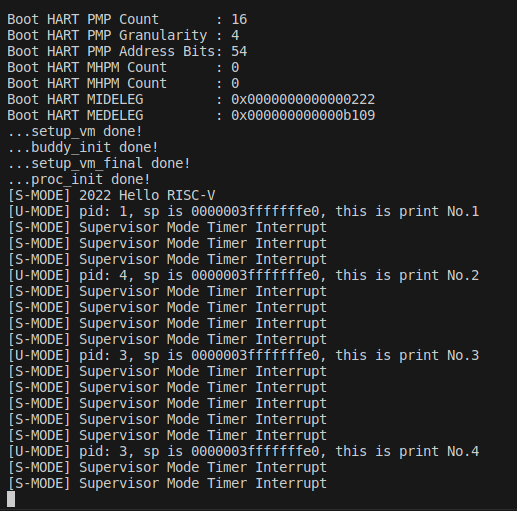
下面具体实现ECALL\_FROM\_U\_MODE的中断处理逻辑。根据手册要求完成即可。



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。

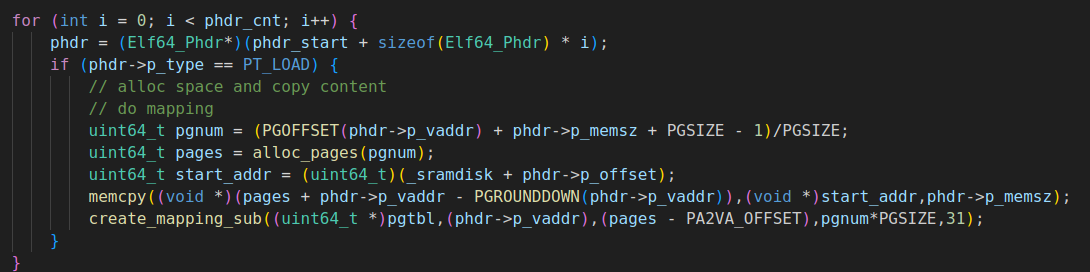
1. 修改head.s和start\_kernel并测试二进制文件

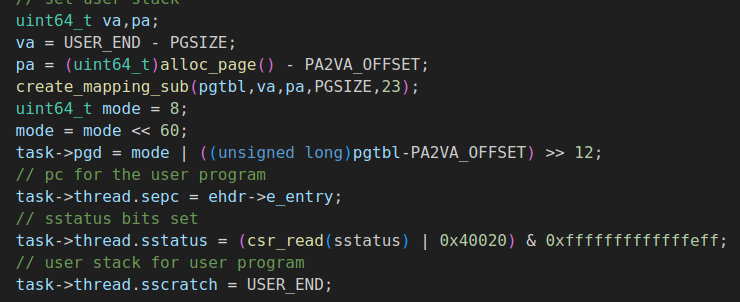
按照要求修改head.s和start\_kernel，就初步完成了该实验，我们可以进行测试。



1. 添加ELF支持

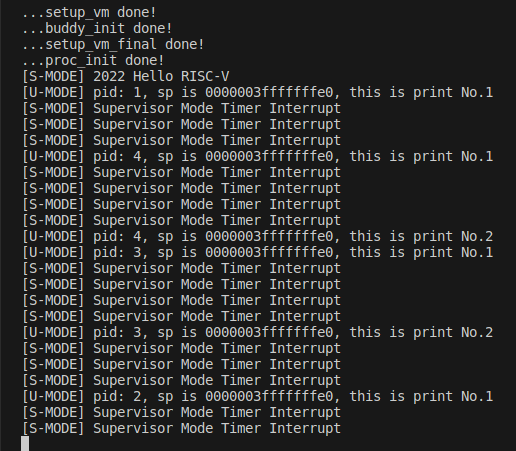
接下来我们要将二进制文件更换为ELF文件，并做一定的修改，继续输出正确结果。首先将upp.s中的payload修改为ELF文件uapp，然后阅读手册并查阅资料完成load\_program函数。





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。



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。

1. 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。