Lab 5

Add Exception handler

- 我们需要添加两个系统调用，get_pid和write。
- 系统调用本身的代码并不复杂，不过有一些需要注意的细节
  - 系统调用的参数和返回值通过寄存器传递的规范，但是我们在Trap返回的时候会恢复寄存器，那么要怎么修改寄存器呢：我们需要修改栈上保存的寄存器
  - Sepc += 4 ?

Context between Kernel & User

- 我们之前也做过寄存器的保存和恢复，不过当时我们实际上没有区分 **用户栈指针(User sp)** 和 **内核栈指针(Kernel sp)**，他们都是`sp`，不过显然，他们肯定不能用同一块区域。
- 从User => Kernel，我们需要
  1. 先保存User Sp到`sscratch`，从`sscratch`加载出Kernel sp。
  2. 然后将sscratch保存到栈里，也就是将User Sp保存到栈里，将sscratch清零。
- 从Kernel => User
  1. 将Kernel sp保存到`sscratch`，将User sp从栈中加载出来
- 从Kernel <=> Kernel (在什么情况下会遇见呢？我们如何区分User => Kernel？)
  1. 我们不需要对sp和sscratch进行任何操作，因为都是用的同一个。
- User <=> User ?

Q & A

- 什么时候会遇见?
  - 我们目前有一个时钟中断，一个ecall异常。
  - 处理ecall异常的时候可能会时钟中断。
- 如何区分?
  - User => Kernel 我们将sscratch清零。
  - 如果sscratch == 0，我们就知道是Kernel => Kernel。
  - 如果sscratch != 0， 此时sscratch是Kernel Sp，是User => Kernel
- User => User ?
  - RISC-V 没有这种情况。

Task create

- Linux: fork & exec
- 我们的实验将其简化，直接在之前task_init的时候把进程设置好。
- 虽然指导告诉大家怎么设置寄存器，还是希望大家好好思考为啥要这么设置：
  - Sepc: 从内核返回用户态执行的地址，应该是User Task's entry point
  - Sstatus: 里面有很多控制bits，决定了sret后的Mode、中断开启等。
  - Sscratch & Sp: User Sp & Kernel Sp
  - Stvec: 用户的页表stvec
  - Ra: __dummy, __dummy需要将sp变成User Sp然后sret
- 为User Task映射好内存，注意每个进程都有自己的内核部分和用户部分。
  - 你也可以让内核部分和用户部分分属两个不同的页表，这不属于实验内容。
- __switch_to 需要把上面提到的CSR也做切换。

ELF

- 我们希望添加对ELF的支持，在实验中，我们可以认为ELF是代码+元数据。ELF的具体细节不做介绍。
- 我们不需要从文件系统加载ELF文件，他直接就在内存中。
- 我们要做的就是从元数据提取需要的信息来设置寄存器和映射页表
  - 主要是 Sepc: Entry point address
  - 根据Program Header 里面的offset, va, flag, type, size等信息映射页表
  - 其他寄存器的设置没有区别。

ELF Example

- ELF有两个Type==LOAD的Section，那么他就分成两块装载到内存中。以第2块为例
- Offset = 0x4000, FileSize = 0x2000, MemSize=0x6fa0: alloc一块大小为0x6fa0的区域，将从[elf.start + 0x4000, elf.start + 0x6000)逐字节拷贝过去，剩下的区域清零。
- VirtAddr = 0xffffffe000203000, Flag=RW : 将上面alloc出来的区域，映射到0xffffffe000203000，大小就是MemSize，权限位是RW，还有U bit别忘了设置

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000001000 0xffffffe000200000 0x0000000080200000
                 0x00000000000021fc 0x00000000000021fc  R E    0x1000
  LOAD           0x0000000000004000 0xffffffe000203000 0x0000000080203000
                 0x0000000000002000 0x0000000000006fa0  RW     0x1000
```