思考题部分：

**思考题1.思考并回答下面的问题：**

**• 内核在保存现场的时候是如何避免破坏通用寄存器的？**

**• 系统陷入内核调用后可以直接从当时的 $a0-$a3 参数寄存器中得到用户调用msyscall 留下的信息吗？**

**• 我们是怎么做到让 sys 开头的函数“认为”我们提供了和用户调用 msyscall 时同样的参数的？**

**• 内核处理系统调用的过程对 Trapframe 做了哪些更改？这种修改对应的用户态的变化是？**

答：

（1）保存现场时新建了一个数据栈结构来存放，将通用寄存器存入。内核使用k0、k1两个寄存器保存用户栈、取出内核栈。

（2）能，没有改变数据。

（3）系统调用中参数的传递依赖于a0-a3参数寄存器和栈。这是汇编决定的，a0-a寄存器不变，且我们从栈中取出和存入保持一致就能够让sys开头的函数认为参数相同。

（4）v0寄存器被更改了，首先v0用来存放参数a0的值，以便从TF\_REG2中取出操作，v0作为系统调用返回值。同时系统调用对EPC+4了，使得调用返回后从下一条正常指令执行。

**思考题2**.**思考下面的问题，并对这两个问题谈谈你的理解：**

**• 子进程完全按照 fork() 之后父进程的代码执行，说明了什么？**

**• 但是子进程却没有执行 fork() 之前父进程的代码，又说明了什么？**

答：（1）子进程在fork函数才产生，保存了fork()之后的状态；

（2）生成子进程不会更改父进程原本的状态信息。

**思考题3.关于 fork 函数的两个返回值，下面说法正确的是：**

**A、fork 在父进程中被调用两次，产生两个返回值**

**B、fork 在两个进程中分别被调用一次，产生两个不同的返回值**

**C、fork 只在父进程中被调用了一次，在两个进程中各产生一个返回值**

**D、fork 只在子进程中被调用了一次，在两个进程中各产生一个返回值**

答：C

**思考题4.如果仔细阅读上述这一段话, 你应该可以发现, 我们并不是对所有的用户空间页都使用 duppage 进行了保护。那么究竟哪些用户空间页可以保护，哪些不可以呢，请结合 include/mmu.h 里的内存布局图谈谈你的看法。**

答：UTOP以下的正常空间可以，UTOP以上的其实在boot\_pgdir中复制了。

**思考题5.在遍历地址空间存取页表项时你需要使用到 vpd 和 vpt 这两个“指针的指针”，请思考并回答这几个问题：**

**• vpt 和 vpd 的作用是什么？怎样使用它们？**

**• 从实现的角度谈一下为什么能够通过这种方式来存取进程自身页表？**

**• 它们是如何体现自映射设计的？**

**• 进程能够通过这种存取的方式来修改自己的页表项吗？**

答：（1）vpt取出对应的页表项，vpd取出对应的页目录项，利用(\*vpt)[]取出。

（2）VPN()操作得到虚拟地址高20位，即物理页号，在\*vpt[]中索引得到。

（3）VPN()/1024大小正好对应着自映射的页目录项的索引位置。

（4）进程应该是不可以这样修改页表项的。

**思考题6.page\_fault\_handler 函数中，你可能注意到了一个向异常处理栈复制**

**Trapframe 运行现场的过程，请思考并回答这几个问题：**

**• 这里实现了一个支持类似于“中断重入”的机制，而在什么时候会出现这种**

**“中断重入”？**

**• 内核为什么需要将异常的现场 Trapframe 复制到用户空间？**

答：（1）当一个中断发生时，还没处理完成又发生中断了，所以要保证一个中断发生时，关闭中断，让系统内核和不再接收以后发生的中断，待当前中断处理完成再接收。

（2）缺页中断处理实质是用户进程自身去完成的，所以需要将异常处理现场复制给用户进程。

**思考题7.到这里我们大概知道了这是一个由用户程序处理并由用户程序自身来恢复运行现场的过程，请思考并回答以下几个问题：**

**• 用户处理相比于在内核处理写时复制的缺页中断有什么优势？**

**• 从通用寄存器的用途角度讨论用户空间下进行现场的恢复是如何做到不破坏通用寄存器的？**

答：（1）不会更改异常运行现场。

（2）通过跳转汇编关键字函数，从而在汇编代码的程度上考虑不破坏通用寄存器。增加nop指令可以避免跳转后栈指针的变化。

**思考题8.请思考并回答以下几个问题：**

**• 为什么需要将 set\_pgfault\_handler 的调用放置在 syscall\_env\_alloc 之前？**

**• 如果放置在写时复制保护机制完成之后会有怎样的效果？**

**• 子进程需不需要对在 entry.S 定义的字 \_\_pgfault\_handler 赋值？**

答：

（1）在系统初始状态下，进程是没有分配页的，也就是首次分配会产生缺页中断，需要处理。

（2） 若写时复制机制已完成，则缺页中断处理时不会达到更新效果，因为写时复制是依赖于缺页中断的。

（3）不需要。

实验难点图示：

1. 这次的实验debug时间是写代码的三倍完全不知道bug在哪里无法寻找。
2. 系统调用的函数，这个函数主要是汇编代码，按照要求理解思路，比较容易，但是之前的错误存在，所以没有得到高分。
3. IPC进程间的同信是进程数据“交流”的方式，这里还要注意sched调度函数的修改，时间片轮询的方式不变，要保证当前进程的状态不为RUNNABLE.
4. Fork()函数的编写以及调用，这是创建父子进程的关键。
5. 缺页中断函数。

STEP

05

缺页中断

page\_fault\_handler

sys\_set\_pgfault\_handler

pgfault

sys\_set\_env\_status

STEP

04

Fork

sys\_env\_alloc

user/fork.c

duppage 函数

STEP

03

IPC

sys\_ipc\_recv

sys\_ipc\_can\_send

STEP

02

Int sys\_mem\_alloc

sys\_mem\_map

sys\_mem\_unmap

sys\_yield

STEP

01

系统调用

user/syscall\_wrap.S 中的 msyscall 函数

syscall.S 中的提示，完成 handle\_sys

体会与感想：

本次lab4作业难度不是特别高，主要是理解机制。之前写的代码一定存在许多bug,所以课下和课上一点也不顺利，一直在寻找Bug。

对于fork()函数的调用以及内部实现，根据代码理解了sys\_env\_alloc在不同的进程可以返回两个返回值：curenv->env\_id为其父进程。将当前的环境中的所有寄存器的状态全部保存在child->env\_tf中，这一步相当于为子进程配置了和父进程完全一眼的进程上下文。在fork结束之前我们不能将子进程状态设置为RUNNABLE，因为我们还将在父进程中为子进程复制一些资源以及处理一些东西。我们首先要运行sys\_env\_alloc()函数，其返回值放在了eax寄存器，在mips中称之为v0寄存器，即regs[2]。然后下一步是将eax寄存器中的值赋给envid。