实验思考题

4.1

- 内核先保存了sp和v0，然后利用这两个和k0k1去保存其他现场，其他的寄存器不被使用
- 可以，因为从用户态调用syscall到调用msyscall再到进入handle_sys,这几个寄存器始终没有改变
- 保存一个没用的sysno参数来占位
- 更改了EPC，使得在用户看来自己跳过了这一句syscall

4.2

- 说明fork之后子进程获得了父进程的几乎全部信息
- 但是子进程栈帧里面保存的pc是父进程的epc，也就是说会从fork那一句话开始执行。父进程中创建了子进程，子进程利用fork的返回值来确认自己的存在。因此子进程不会执行fork之前的代码

4.3

- A错误，在父进程中调用1次，返回子进程id
- B错误，子进程没有调用fork
- C正确，子进程在sys_env_alloc中被创建，并从这里开始执行
- D错误，父进程必须调用了fork，要不然子进程从哪儿生出来的呢

4.5

- 作用是快速访问到自己页表里面的东西，顺便找页
- 其实这两个数组的内容就是一一映射到了UVPT那一部分
- 自映射机制体现在entry.S中，vpd是UVPT+(UVPT>>12)*4
- 不能。因为这一部分在USTACKTOP以上，ULIM一下，只有内核态能写

4.6

- 嵌套异常/中断
- 发生异常中断时，内核会把栈复制到异常栈，但嵌套的话，异常栈还没有准备好，就会破坏上一个异常栈

4.7

- 内核的工作减轻了，符合设计思想。另一方面，可能有的用户比较牛逼，有专属的处理函数
- 保存到了自己的栈帧，只用那几个t来搞事情

4.8

- 因为子进程也需要这一步，而alloc出来一个0后，即子进程调用fork时，直接返回了。
- COW建立完成后，所有可写的都变成了COW，但是此时子进程还没跑起来。另外，COW建立是在alloc之后，和上一题可以构成**循环论证**
- 不用，和父进程设置的一样(在fork里面)

实验难点与心得体会

- 不得不说，lab4相比lab3，玄学程度高了不止一个档次。我现在基本能全部掌握lab2，能大致掌握lab3，但绝对不敢说了解了lab4。而各种玄学bug也为lab4增添了一些神秘色彩
- 首先，我在lab4中使用了简易的对拍器。具体原理为，输出重定向，writef内部含正则，python抓输出，然后肉眼比对。可以说很low，但是凑活能用。如果能解决gxemul自动关闭重启的问题和测试样例（用户.c）的自动生成问题，那么一个批量化运行的脚本就可以正式部署了。如果能解决远程服务器无法联网且没有标程导致的无法对比或没有SPJ问题，那么对拍器就可以正式上线了。
- 一开始的debug中，面对“硬件”级别，也就是lab2lab3（他们对于用户来说和硬件没啥区别）的bug，可以通过各种方式，包括课上测试提供的思路，dump，手动查，printf大法，debug函数关钟+printf+断点等等，不择手段进行debug。然而，当问题来到user层面，事情就变得有趣了起来。
- 首先，不知道是gxemul神奇的双钟（实中断，虚主频）导致的不一定可复现的线程安全bug，还是真的问题出在了编译器上，我遇到过一些匪夷所思的bug，如下：
    - 多出了一句writef或者syscall_getenvid导致不同的结果，这毕竟都syscall了，出bug很正常
    - writef更改参数导致的bug，这毕竟改了栈，出bug还算正常
    - 汇编内部增加一句move指令，这毕竟不是标准MIPS指令，出bug也说得过去
    - 汇编内部增加一句addu t0,t0,$0，这毕竟改变了汇编函数的大小，万一卡在了页的边上，我认了
    - 汇编内部将一句addu $0,t0,t0改成addu t0,$0,t0，这可是什么都没有改变，连loadelf都不会变的改动，仍然会出bug。且两边都能复现4-5次（固定没bug，固定有bug）

  看到最后一步，我彻底惊呆了。我无法理解发生了什么。也许是4-5次次数不够，也许是编译器的乱序处理出现了bug，也许是OS没有完备的鲁棒处理导致中途有return<0的函数却没有被报错。总之，这样的错误令我匪夷所思。
- 总的来说，lab4-1还是比较清晰地。掌握了syscall这类行为API-syscall-msyscall-trap-sys-return 这一全过程，以及学会部分API的使用，大概差不多。
- lab4-2主观来看也还行，难点集中在缺页相关的各种事情，但各种不明原因的玄学bug令4-2显得更难了

实验建议

- 第一，强烈建议增开答疑课，负责解决各种玄学bug
- 第二，强烈建议公开测试方法，究竟是抓输出，还是用特殊的user程序进行测试，以及，是否会替换一部分文件
- 第三，强烈建议课上测试时，公开各个测试点的错误原因，至少是**部分**测试点的**大致**错误原因，同时提供一些样例输入与输出
- 第四，建议公开部分std思路，比如明确部分注释，而不仅是"set xxx to a suitable value"，或者公开评测机的大致原理。明确所填代码**正确**的标准。

实验心得

- 总的来说，到了lab4这一步，可以说真正明确了OS为用户服务这一用途，就好像计组从P6到P7一样，从单纯的CPU/OS到了服务端。但是，黑盒的评测系统和各种难以名状的bug带来了不太舒服的体验
- 总的来说，建议公开测试方法，这样既有利于debug，有利于整个课程的理解与学习，又有利于大家自己创造SPJ与GEN从而搭建有效的评测系统。如果担心靠纯暴力输出卡过评测，可以公开标准的同时，课下设置20组用户程序进行测试，比如测试fork或者ipc，或者杂交，顺便测加载镜像，写时复制等等各种问题。公开标准的同时，公开5-6组的期望输出，我认为这样可能更有利于课程的学习。
- 以及，由于OS没有计组那样重考的机制，建议每个lab结束后，公开思考题的答案，明确理解的同时消除我们的错误认知，更好的进行学习。我个人认为，公布了考完的lab的思考题答案，不会存在对后续恶劣的影响。
- 主观来说，对于OS的学习效果与体验远不如计组的原因，除了以上客观原因，还有我自己的主观原因。试想，如果对于gcc的使用像ISE一样明确，对于gxemul的理解像mars一样稍微彻底一些（毕竟mars源码裸奔，甚至

可以自己改~~，评测系统和网站像cscore一样环境友好~~），那学习OS将会更有乐趣，更有收获。我现在的OS程序，可以说是一步一个坑，自己造的样例都能把自己卡住，甚至还带着有已知未定位bug的程序去参加课上测试。如果我对于OS的理解再多些，练习再多写，我可能也能像去年一样做到至少能尽可能1A弱测，自信面对中测，敢于挑战强测吧