思考题

4.1

- 先保存`sp`和`v0`，再利用这两个寄存器和`k0 k1`去保存现场，其他寄存器不被使用
- 可以，保存参数的寄存器没有被改变
- 利用一个没有实际意义的`sysno`参数来占位，并且我们在栈中为每个参数都开辟了空间，并放入正确的位置
- 将`EPC`的值加`4`并存回原来的位置，这样用户就感觉跳过了`syscall`指令

4.2

- 说明在`fork()`之后子进程和父进程共享一段代码
- 子进程在`fork()`之后才被产生，并且其PC值为父进程调用`fork()`之后的位置，所以只能执行之后的代码，不会执行之前的代码

4.3

- C正确，父进程调用`fork()`并在`sys_env_alloc()`中创建子进程

4.4

- 比`USTACKTOP`小的用户空间页可以保护，在`USTACKTOP`之上的用户空间页不可以被保护，这些用户空间页是所有进程共享的，不是某一个进程私有的

4.5

- `vpd`宏是指向用户页目录的指针，`vpt`宏则是指向用户页表的指针，作用是可以快速访问自己页表的内容，也可以找页
- 这两个数组中的内容就是映射到`UVPT`那一部分
- 在`entry.S`中，有这样的语句`(UVPT+(UVPT>>12)*4)`
- 不能，这一部分是在`USTACKTOP`之上的用户空间，不是进程私有的，只有内核态可以写

4.6

- 中断嵌套或者异常嵌套，发生缺页错误的页面是正常堆栈的页面
- 当出现异常嵌套的时候内核栈还没有完全准备好，此时前一个异常栈会遭到后一个异常的破坏，所以需要将异常的现场复制到用户空间

4.7

- 符合微内核的思想，减轻了内核的负担
- 在`MIPS`的通用寄存器中，`tn`寄存器作为临时寄存器可以被随时使用，所以可以用这几个寄存器来恢复现场

4.8

- 因为在创建子进程的时候会触发缺页中断，需要提前设置缺页中断处理函数
- 缺页中断将不会正常执行
- 不需要，和父进程相同

实验难点

- 在本次 lab4 的实验中我认为较难的函数之一是 sys_env_alloc，需要填写的语句并不难，难的是需要理解为什么这个函数在父子进程中返回的是不一样的值，我仔细阅读指导书和代码之后我对这个问题有了一些理解，首先看较为关键的代码：

```
e->env_tf.pc = e->env_tf.cp0_epc;
e->env_tf.reg[2] = 0;
return e->env_id;
```

在调用 syscall_env_alloc 之后，子进程的 PC 值被设置成发生异常处下一条指令的 PC 值也就是 PC+4，然后作为携带返回参数的 $v0 寄存器被设置成了 0，所以子进程就感觉自己并没有执行完这些代码而直接得到返回值是 $v0 的值也就是 0；而父进程执行的是完整的代码，所以返回值是子进程的进程号

- 在此次的实验中另一个让我感到意外的函数是 sys_mem_map()，这个函数其实是比较简单的，就是将原进程需要映射的物理页映射到目标进程相应的位置上，过程大概就是通过 page_lookup 找到相应的页，在通过 page_insert 把得到的页插入相应的位置上。随着debug过程的不断深入，我发现我这么写其实有问题（无穷无尽的 pageout）；修改之后变成利用 page2pa 函数和 page_lookup 的参数之一 *pte 得到相应的页，再插入相应的位置，这样修改可以顺利通过评测。
  但是交流之后发现有很多同学的 sys_mem_map() 都是前一种写法且并没有发现什么问题，这着实让我有一些疑惑，不太清楚是哪里出了问题。

实验心得和体会

- 记得助教在 lab3 中说 lab3 应该是最难的实验了，当时顺利通过 lab3 课下之后着实有点庆幸，但到了 lab4，发现事情并不是那么的简单，做了整整三天，一天写代码，两天 debug。~~我还是太年轻~~
- 不得不说，OS课程各种玄学问题层出不穷，最重要的是 debug 手段实在是有限，目前我采用的手段主要是：
  - 瞪眼法　　瞪 瞪 瞪 瞪 瞪 瞪 我就不信瞪不出来
  - 把 printf/writef 撒的遍地都是　实在是瞪不出来就只能靠输出信息来调试了
  - 其实还有几种调试方法，比如断点调试法，但我还没试过，以后可以试一下
  - ~~debug愉悦指数-2147483648，不能再小了，再小就溢出了~~
- 总的来说，这次的 lab 难度很大，需要花费很多的时间和精力。通过这次的实验我也基本上掌握了系统调用的基本过程，从用户态陷入内核态的过程、从用户函数如何一步一步调用内核函数的过程以及参数传递的过程。
  但是对于 fork() 过程的还不是很清晰，需要考虑底层的机制和问题比较多也比较麻烦，尤其是处理子进程缺页中断的一系列操作让我不是很理解，需要进一步阅读代码和指导书。

EXTRA

此次的 extra 有一定的难度，但也都是基于课下，并没有涉及到全新的知识。但是我却花费了很多时间，主要在进阶测试的第一部分，就是因为我在写函数时没有加入那个不起眼、没有什么存在感的参数 int sysno，浪费了一个白天的时间，给了我一个很大的教训。