Lab4实验报告

17373452 单彦博

一、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?**

  内核会把除了sp以外的其他寄存器保存到栈里，并做好维护。

- **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?**

  不可以。陷入内核态之后，寄存器的值可能会发生变化，此时应从内核态的栈sp里面去取各个寄存器的值。

- **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?**

  先从sp里面将a0-a3取出，然后找到用户态的栈指针，从用户态的栈里面取出其他的参数，放到内核态的栈里面。

- **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?**

  将epc的值加4，保证系统调用返回后用户态进程可以从下一条指令继续执行。将系统调用的返回值存到v0里面，供用户态函数使用。

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?**

  说明了子进程很可能与父进程共享了内存空间，导致他们的程序代码都是相同的。

- **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?**

  说明fork产生的子进程的pc被改成了父进程fork之后的pc值。

3、关于fork 函数的两个返回值，下面说法正确的是：（C）

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

4、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。

在USTACKTOP以下的页，UTEXT以上的页可以被保护，而在USTACKTOP到UTOP之间的位置不能被保护，即不能被duppage，原因是如果我们将[UXSTACKTOP-BY2PG,UXSTACKTOP]这块区域duppage到子进程，那么父进程与子进程的错误栈就一样了。但是由于两个进程的调度时机不同，子进程应该为其分配一个新的错误栈。所以[UXSTACKTOP-BY2PG,UXSTACKTOP]的区域不应该被保护，而[USTACKTOP,USTACKTOP+BY2PG]的区域是无效内存，因此也不应该被保护。此外，在这个区域中我们对于只读的页不能加上PTE_COW权限，而应该只给可写的页或者原本是PTE_COW权限的页加上PTE_COW权限。因为我们不能将原本不可写的页的权限改成可写。

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?**

  二者是在user/entry.S里面定义的。vpd宏是指向用户页目录的指针，即*vpd=7fdff000=(UVPT+(UVPT>>12)*4)，vpt宏则是指向用户页表的指针，即*vpt=7fc00000=UVPT。通过这二者，我们可以得到一个虚拟地址va的页目录项以及其页表项，用法就是(*vpd)[(VPN(va)>>10)]即得到对应的页目录项，(*vpt)[VPN(va)]即得到对应的页表项。

- **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?**

  这两者出现的背景是为了在用户地址空间中实现对内核中相应地址的访问，如果要我在lab2中实现同样的功能，我可以采用UVPT+VPN(va)来代替(*vpt)[VPN(va)]。

- **它们是如何体现自映射设计的?**

  可以看到vpd的位置是在UVPT和UVPT + PDMAP之间的，这表明vpd是vpt的其中一个，所以这里体现了自映射的设计。

- **进程能够通过这种存取的方式来修改自己的页表项吗?**

  不可以，这只是一个访问的方式，用户态不可以修改页表项。

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”?**

  在发生时钟中断的时候。

- **内核为什么需要将异常的现场Trapframe 复制到用户空间?**

  保存寄存器的值防止这些值被破坏。

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?**

  可以减少进程陷入内核态后操作系统内核的工作量，加快内核的运行速度。

- **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?**

  通过将通用寄存器的值压入栈中，然后在使用时取出，可以避免破坏通用寄存器中所存储的值。在取出的时候先取出除sp之外的其他寄存器，然后在跳转返回的同时，利用延时槽的特性恢复sp。

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?**

  因为不能让子进程进行set_pgfault_handler，同时可以处理在alloc过程中所实现的缺页错误。

- **如果放置在写时复制保护机制完成之后会有怎样的效果?**

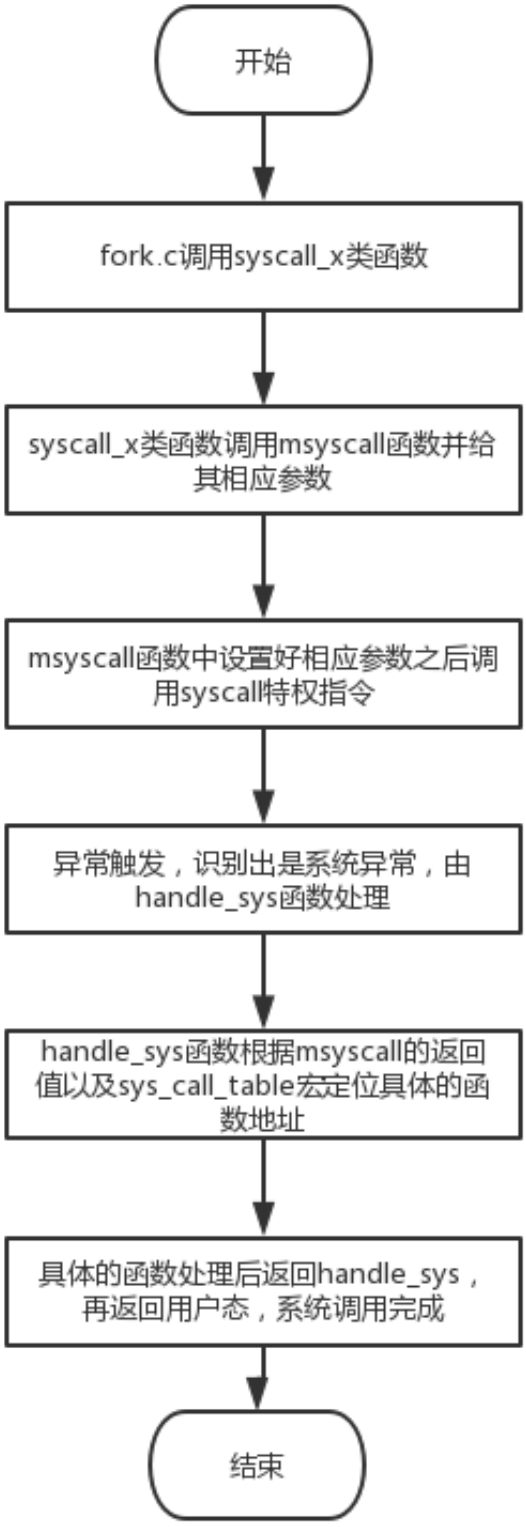
  会导致写时复制机制无法正常运行，缺页错误无法处理。

- **子进程需不需要对在entry.S 定义的字__pgfault_handler 赋值?**

  不需要。因为在父进程中的值与子进程的相同，所以不需要重新设置。

二、实验难点图示

本次实验比较难理解的地方就是系统调用的流程，因为涉及到用户对内核态的调用，比较麻烦。



三、体会与感想

本次实验难度较大，课下为了理解syscall的各种调用关系，以及fork的流程，我花费了很多时间，课下我用的时间大概有十几小时。

我在两次的课上测试均未通过基础测试，分析原因，可能是我对整个流程的理解还不是很透彻，当然也有不细心的情况出现，譬如lab4-1-exam的时候我把原代码中jalr的寄存器给修改了，但是一直没发现，导致未通过。在lab4-2-exam中，要实现线程的fork，但是我不是很清楚怎么可以让bss段共享，导致也没有完成测试。我认为，在理解代码方面多下功夫是最重要的。在第一次课上测试之后，我又仔细分析了系统调用流程，对这个流程的理解又更透彻了一点，同时，对整个机制的精妙与复杂不由得发出感叹。