

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 操作系统 |
| 姓 名： | 董佳鑫 |
| 学 院： | 计算机学院 |
| 系： | 计算机系 |
| 专 业： | 计算机科学与技术 |
| 学 号： | 3210102181 |
| 指导教师： | 寿黎但 |

# Lab 6: RV64 fork机制

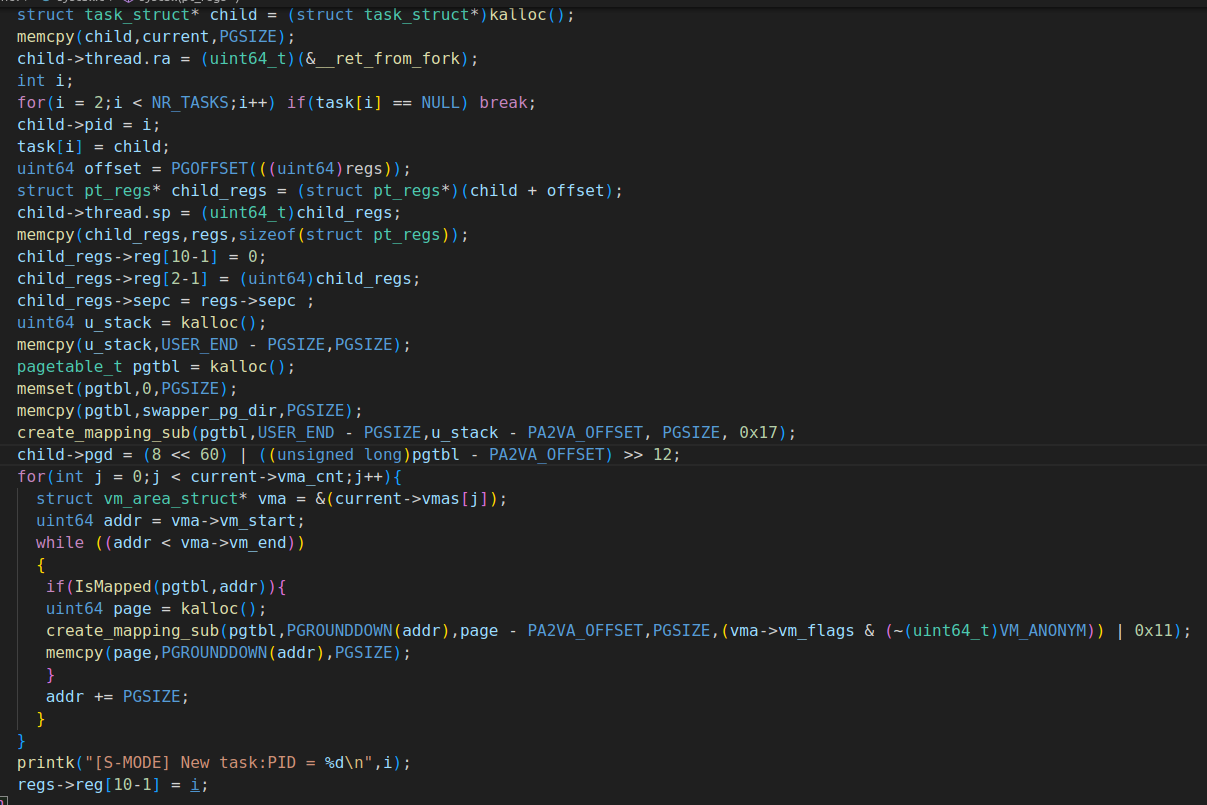
实验目的

1.为 task 加入 fork 机制，能够支持通过 fork 创建新的用户态 task。

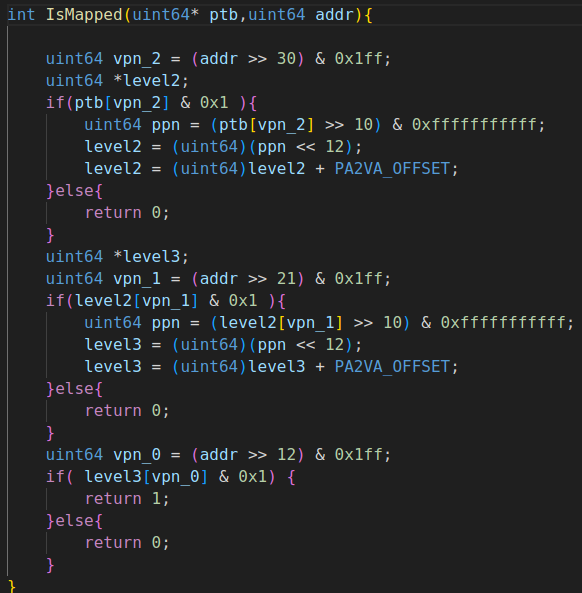
实验过程和操作步骤

1.添加fork系统调用

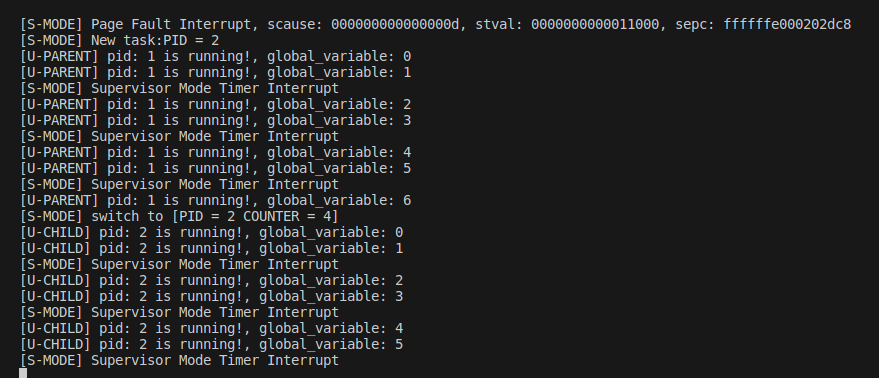
在syscall.c中添加新的系统调用fork。

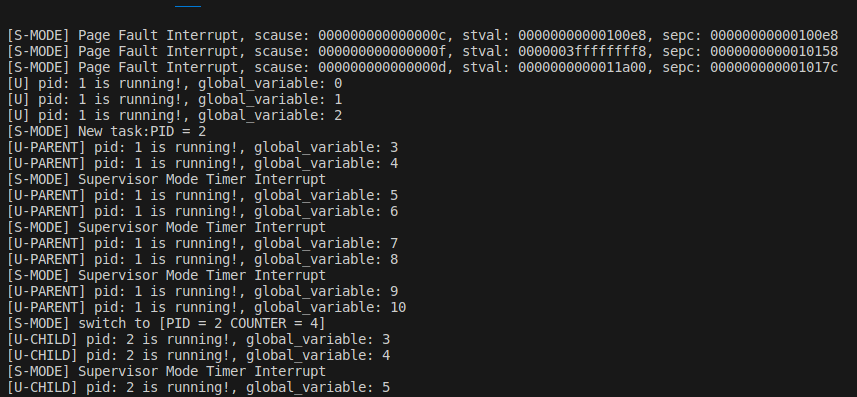


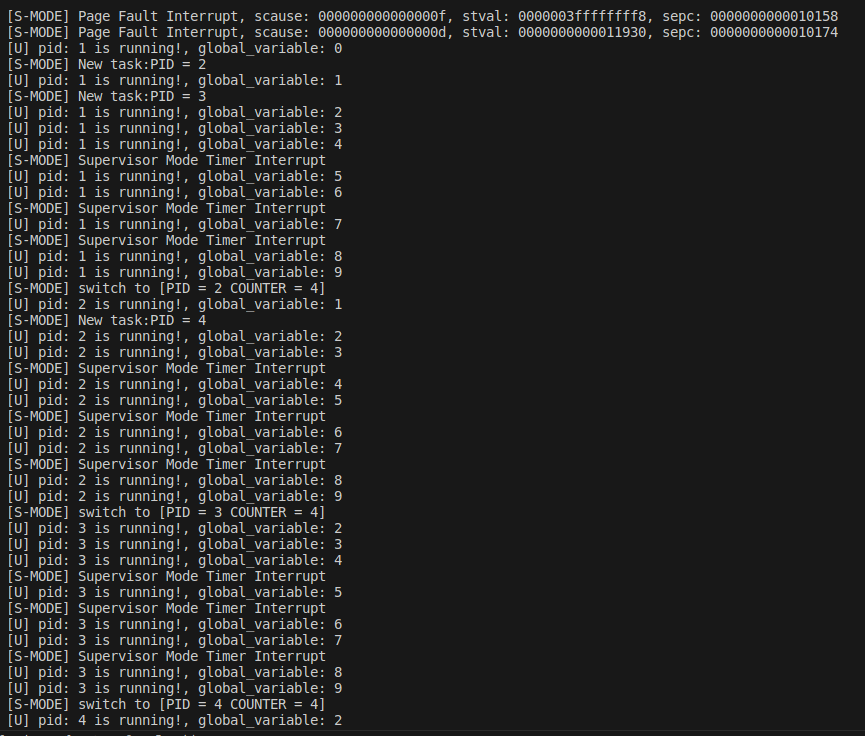
按照实验手册的逻辑完成fork系统调用。其中值得注意的地方有：关于子进程thread.sp的值应该按照父进程pt\_regs的位置进行设置，这在下面的思考题会具体阐述；然后我们要对一块新申请的内存作为子进程的用户态栈，并且把这个栈建立映射，然后拷贝父进程的内容。在设置好必要的变量后，申请并建立对应的页表，这和lab3时建立页表映射类似。然后要根据父进程VMA的映射情况，子进程也随之进行映射。这里会调用函数IsMapped，这个函数的作用主要就是查看目标地址是否能根据三级页表找到对应的物理地址，如果找到了就说明在父进程中已经建立起了页表，此时子进程也要随之建立页表。IsMapped函数实现如下：



完成上述代码，并且完成必要的修改以后就能直接运行测试，下面是分别运行3个main函数后的结果：







思考题

1. 参考 task\_init 创建一个新的 task，将的 parent task 的整个页复制到新创建的 task\_struct 页上。这一步复制了哪些东西?

每次创建task的时候，都会申请一个page，这个page中包括各种构成struct\_task的变量，以及内核态栈的数据。因此我们在复制的时候，会把这个page中的所有数据全部复制到新的task中，包括task\_struct和pt\_regs结构体中的数据。

1. 将 thread.ra 设置为 \_\_ret\_from\_fork，并正确设置 thread.sp。仔细想想，这个应该设置成什么值？可以根据 child task 的返回路径来倒推。

thread.sp应该利用参数 regs 来计算出 child task 的对应的 pt\_regs 的地址，也就是child + PGOFFSET(regs)。Child task的返回路径是\_\_switch\_to->\_\_ret\_from\_fork->user program，而在\_\_ret\_from\_fork（traps）中，所用到的sp应该是内核态的指针，也就是指向各自结构体pt\_regs的指针，因此应该利用参数的regs来计算child task的对应的pt\_regs地址。

3.利用参数 regs 来计算出 child task 的对应的 pt\_regs 的地址，并将其中的 a0, sp, sepc 设置成正确的值。为什么还要设置 sp?

因为我们在返回到\_\_ret\_from\_fork(\_traps)中，需要restore寄存器的值，这就包括了sp。因此我们的regs->reg[2](sp)需要设置为指向pt\_regs的指针，这才能保证restore以后还能正常找到正确的sp。

讨论心得

本次实验遇到的问题是，在设置child task的时候，有很多变量的设置需要慎重考虑。我在这里遇到的问题是，对于父进程和子进程的程序流不太清楚，在认真学习和思考以后，搞懂了其中的原理后也顺便复习了操作系统的期末知识。

本次实验是我们完成的最后一个OS实验，回想这一个学期的实验旅程，实在是颇为感慨。难度实在是不小，不过收获也颇丰。从内核引导，到时钟中断处理，到虚拟内存创建，到用户态线程，再到缺页异常处理和fork机制。自己在这一个学期一步步建立了一个操作系统，也是蛮有成就感的一件事。