Lab 4 Report

21371295 张昊翔

思考题

Thinking 4.1

- 使用SAVE_ALL宏，将**通用寄存器**的值存在结构体TrapFrame中，系统调用结束后又会重新将这些值放回通用寄存器。
- 可以，用户态和内核态**共用**一套通用寄存器。
- 前四个参数存放在$a0-$a3**寄存器**，后两个参数按存入**栈帧**中的8字节空间。

```
    sysno = tf->regs[4];    # $a0
    func = syscall_table[sysno];

    arg1 = tf->regs[5];     # $a1
    arg2 = tf->regs[6];     # $a2
    arg3 = tf->regs[7];     # $a3

    arg4 = *(u_int *)(tf->regs[29] + 16);   # $sp + 16 bytes
    arg5 = *(u_int *)(tf->regs[29] + 20);   # $sp + 20 bytes
```

- `tf->cp0_epc += 4`：EPC + 4，返回时执行syscall的后一条执行，避免再一次执行**陷入内核**的指令。

Thinking 4.2

检查通过`e = &envs[ENVX(envid)]`取出的**进程结构体**e是否正确。

Thinking 4.3

mkenvid()函数用于生成新进程结构体的ID: `e->env_id = mkenvid(e);`

```
u_int mkenvid(struct Env *e) {
    static u_int i = 0;
    return ((++i) << (1 + LOG2NENV)) | (e - envs);
}
```

或运算生成的返回值显然不为0，因此envid2env()函数在输入ID为0时，不再寻找ID对应进程，直接返回**当前进程**。

Thinking 4.4

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
父进程返回**子进程**的envid，子进程返回0。

Thinking 4.5

kuseg是用户态下唯一可用空间。

- ULIM-UVPT: **进程页表**，对应唯一的进程，显然不能复制
- UVPT-UTOP: 在 `mips_vm_init` 中建立映射，已对所有进程共享
- UTOP-USTACKTOP: **异常处理栈**，所以父子进程不需要共享这部分的内存，无需映射

因此，只需映射USTACKTOP以下空间，即USTACKTOP-0。

Thinking 4.6

vpt和vpd分别是指向**用户空间**中的页表和页目录**首地址**的指针，可通过**索引**获取虚拟地址对应页表项或页目录项的内容。

```
#define vpt ((volatile Pte *)UVPT)
#define vpd ((volatile Pde *)(UVPT + (PDX(UVPT) << PGSHIFT)))
```

- PDX(va): 31-22 页目录索引
- PTX(va): 21-12 页表索引
- VPN(va): 31-12 虚拟页号

vpt[VPN(va)]: 通过**虚拟页号**获取页表项（不经过页目录，直接访问），**遍历**页表使用BY2PG
vpd[PDX(va)]: 通过**页目录索引**获取页目录项，**遍历**页目录使用PDMAP

Thinking 4.7

因为页写入异常是在**用户态处理**，在用户空间使用系统调用syscall_set_trapframe()，可以恢复事先保存好的现场，使得用户程序恢复执行。

Thinking 4.8

在用户态处理页写入异常相比于在内核态处理的主要优势是**安全性。**
当一个用户程序试图访问它没有权限的内存区域时，如果在内核态处理这个异常，那么这个用户程序可能会因为内核态的特权级别过高而能够执行一些恶意的操作，例如读写内核数据结构，篡改内核代码等等，导致系统的不稳定和不安全。如果在用户态处理这个异常，内核将以用户态的权限运行，不会暴露内核的特权级别。

Thinking 4.9

- 在syscall_exofork()之前，可以**统一**设置父子进程的页写入异常的异常处理函数。
- 写时复制保护位于duppage()函数中，在duppage()之前，可以处理duppage()中遇到的**页写入异常。**

难点分析

系统调用

- 用户态
  - user/lib/syscall_lib.c: `syscall_*()`
  - user/lib/syscall_wrap.S: `msyscall`

---------- syscall ---------->

- 内核态
  - kern/genex.S: `handle_sys`
  - kern/syscall_all.c: `do_syscall()`
  - kern/syscall_all.c: `sys_*()`

4.9 sys_exofork()

`e->env_tf = *((struct Trapframe *)KSTACKTOP - 1);` **寄存器状态**保存在KSTACKTOP以下的一个`sizeof(TrapFrame)`大小的区域。
将KSTACKTOP之下的Trapframe拷贝到**当前子进程**的env_tf中，以达到**保存进程**上下文的效果，复制一份给子进程。

4.10 duppage()

写时复制保护: 设置**写时复制位**PTE_COW，取消可写位PTE_D。若此后父或子进程需要写该页面，则触发页写入异常，**取消共享**，分配新的物理页面。
先将页面映射到子进程中，避免在两条系统调用指令之间，**父进程修改了该页面。**

实验体会

系统调用的实现让我对操作系统中**内核态和用户态**各自的职责有了更深的体会。在这次实验中，我主要关注了IPC和fork()在操作系统中的执行**流程**，具体功能的实现帮助我理解掌握了操作系统的结构设计。