# 实验报告

学号：15051415　姓名：冯冠玺

实验题目：Linux 内核编译及添加系统调用

## 1、实验要求：

（1）添加一个系统调用，实现对指定进程的 nice 值的修改或读取功能，并返回进程最新的 nice 值及优先级 prio。建议调用原型为：int mysetnice(pid_t pid, int flag, int nicevalue,void __user *prio,void__user *nice);

参数含义：

pid：进程 ID。

flag：若值为 0，表示读取 nice 值；若值为 1，表示修改 nice 值。

Prio、nice：进程当前优先级及 nice 值。

返回值：系统调用成功时返回 0，失败时返回错误码 EFAULT。

（2）写一个简单的应用程序测试 （1）中添加的系统调用。

（3）若程序中调用了 linux 的内核函数，要求深入阅读相关函数源码。

## 2、项目设计方案：

（1）本实验总体设计思路；

A)修改 nice 值部分

首先，我发现 Linux 内核中 set_user_nice()函数可以修改指定进程的 nice 值，但是该函数传入的参数是一个 task_struct 结构体，因此如何找到指定进程对应的 task_struct 结构体便成为了重点。

我首先通过 find_get_pid()函数找到指定进程号的 struct_pid 结构体,再通过 pid_task 函数获取到 task_struct 结构体。进而解决了问题。

B)函数返回值部分

copy_to_user 函数可以将数据块从内核空间转换到用户态。但需要注意的是该函数的参数部分。该函数一共需要三个部分一共是用户空间的进程地址，一个是内核空间的源地址，还有一个是需要拷贝的数据长度。针对这三个参数：

void __usr *to：我用(int *)prio 来传递

const __void from：我用&task->prio 来传递

unsigned long n：我用 sizeof(task->prio)函数来传递

（2）本实验中主要函数的接口设计；

主要函数为服务例程：

SYSCALL_DEFINE5(mysetnice, pid_t, pid, int, flag, int, nicevalue, void __user *, prio, void __user *, nice)

其中，mysetnice 是我写的函数名，pid 是用户输入的进程号，flag 是“读取或改变指定进程 nice 值的”的标志，nicevalue(是用户要改变的 nice 值，仅在 flag==1 时有效)，__user *prio 是我要将内核空间中的 prio 传递到用户态的一个地址，__user *nice 同理。

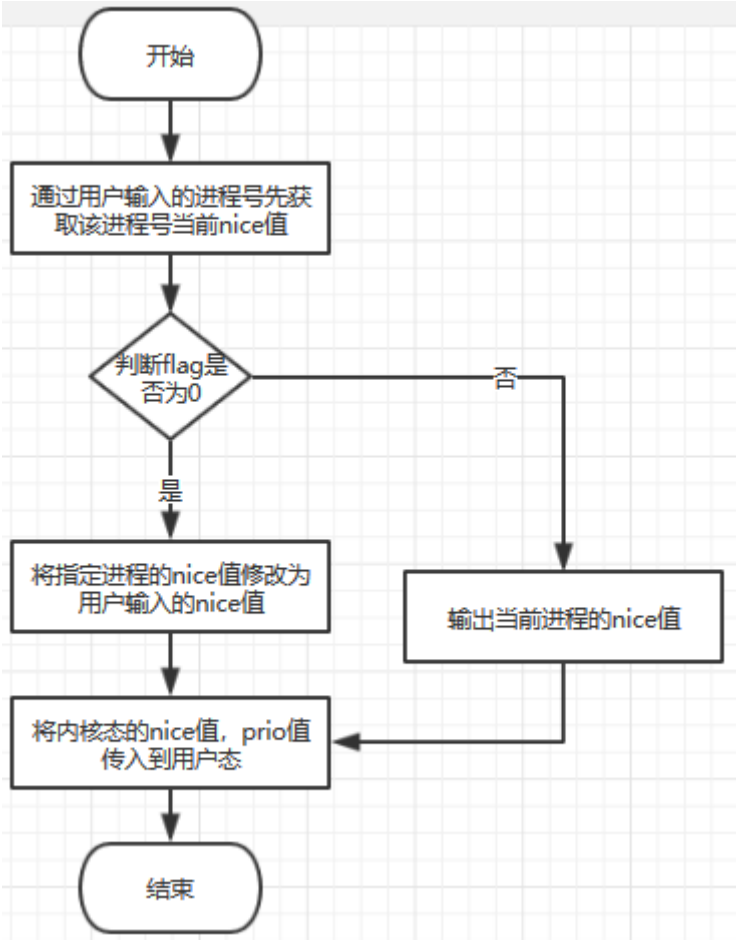
（3）主要函数的程序设计思路，并画出程序设计流程图。

Step 1:由用户指定的进程号，获取对应的 task_struct 结构体，再由 task_nice 函数计算指定进程的 nice 值存放到 nicenow 变量中

Step 2:判断 flag 的值，如果 flag 为 0，则仅将当前 nice 值（nicenow）输出，跳到第 4 步

Step 3:如果 flag 值为 1，根据用户输入的 nice 值，将进程的 nice 值进行修改

Step 4:使用 copy_to_user 函数将 nice 值，prio 值传入到用户空间




（4）对设计方案的创新性分析

传统的，根据用户输入的进程号寻找指定进程的思路是使用一个 for 循环，从全部进程中搜索，匹配进程号。但是这样时间复杂度较高。其实，linux 自身就提供了 find_gety_pid 这一函数，可以直接进行调用。

## 3、项目实现过程：

（1）详细记录项目实现过程中遇到的问题、原因及解决方法（可用截图加文字说明的方法进行编写）

问题 1：一开始没有注意到要有返回值，所以写的函数中只有三个参数

解决：经别人提醒，才发现这一错误，同时深入研究了 copy_to_user 这一函数，得以解决。

（2）分析程序运行结果

验证程序运行结果最关键的事情就是查看是否能将内核态的数据块传入到用户态。在测试代码中我设置了 int prio,nice 值。在 flag=0 时，比较服务历程的输出和 dmesg 的输出，发现结果一致，证明传值成功。另外，置 flag=1 时，通过 top -p 观察 nice 值是否改变，结果证明 nice 值的确有变化，说明程序正确。

（3）分析程序实现中的创新性；

传统的，根据用户输入的进程号寻找指定进程的思路是使用一个 for 循环，从全部进程中搜索，匹配进程号。但是这样时间复杂度较高。其实，linux 自身就提供了 find_gety_pid 这一函数，可以直接进行调用。另外，在传入数据块大小这一参数时，可直接使用 sizeof 函数进行传值。

## 4、对项目的进一步思考：

（1）分析所实现的程序的性能，包括优缺点；

缺点：程序健壮性不够

例如：默认 flag 的值只能为 0 或 1，对用户输入的 nice 值没有进行判断。

（2）考虑的改进思路；

增强健壮性，充分考虑用户输入的可能性，例如 flag 值不为 0 和 1 时。nice 值超出 linux 时，直接报错。

## 5、参考文献（含阅读书籍、论文、网络资源等）

1）操作系统课程实验指导书

2）操作系统实验 1 经验分享：
http://www.th7.cn/system/lin/201711/233811.shtml

3）Linux 源码网站：
http://elixir.freeelectrons.com/linux/v4.14.1/source/kernel/sched/core.c#L3774

## 6、程序完整代码

系统服务例程部分：

```
SYSCALL_DEFINE5(mysetnice, pid_t, pid, int, flag, int, nicevalue, void __user *, prio, void __user *, nice)
{
    //set_user_nice可以修改进程nice值但是传入的参数是task_struct结构体和要修改的值
    //使用find_get_pid函数找到指定进程的struct pid再通过struct pid找到task_struct
    struct pid * mypid;
    struct task_struct * task;
    int nicenow;
    int newnice;
    mypid = find_get_pid(pid);  /* 返回的是struct pid */
    task = pid_task(mypid, PIDTYPE_PID);  /* 返回的是task_struct */
    nicenow = task_nice(task);/* 获取进程当前nice值 */
    if(flag == 1)
    {
        set_user_nice(task, nicevalue );/* 修改进程nice值 */
        printk("原nice值: %d  修改后nice值: %d\n", nicenow, nicevalue);
    }
    else if(flag == 0)
    {
        printk("该进程的nice值为%d\n", nicenow);
    }

    /*注意这里的copy函数是执行成功返回0不成功返回1*/
    if (copy_to_user((int*)prio, &task->prio, sizeof(task->prio)))
            return EFAULT;
    newnice=task_nice(task);/*重新获取nice值*/
    if (copy_to_user((int*)nice, &newnice, sizeof(newnice)))
            return EFAULT;
    return 0;
}
```

测试代码：

```
1   #define _GNU_SOURCE
2   #include <linux/unistd.h>
3   #include <sys/syscall.h>
4   #include<stdio.h>
5   #define __NR_mysyscall 333 /*系统调用号根据实验具体数字而定*/
6   int main()
7   {
8       int nice;
9       int prio;
10      syscall(__NR_mysyscall,4,1,0,&prio,&nice); /*或 syscall(333) */
11      printf("nice:%d\n",nice);
12      printf("prio:%d\n",prio);
13      return 0;
14  }
```