# 操作系统课程设计报告

1. 实验2：
   1. 实验题目：linux内核模块编程

学号及姓名：15058317 李秀峰

2、 项目设计方案：

（1）本实验总体设计思路；

设计两个模块，一个带参数，一个不带参数，先根据书上的指导，将不带参数的模块写出来，直接使用for\_each\_process函数，列出所有进程的信息；带参数的模块，输入进程的pid，然后使用pid\_task函数根据pid找到进程，接着list\_for\_each遍历分别输出该进程的家族信息，包括父进程、兄弟进程和子进程的的程序名、PID号。

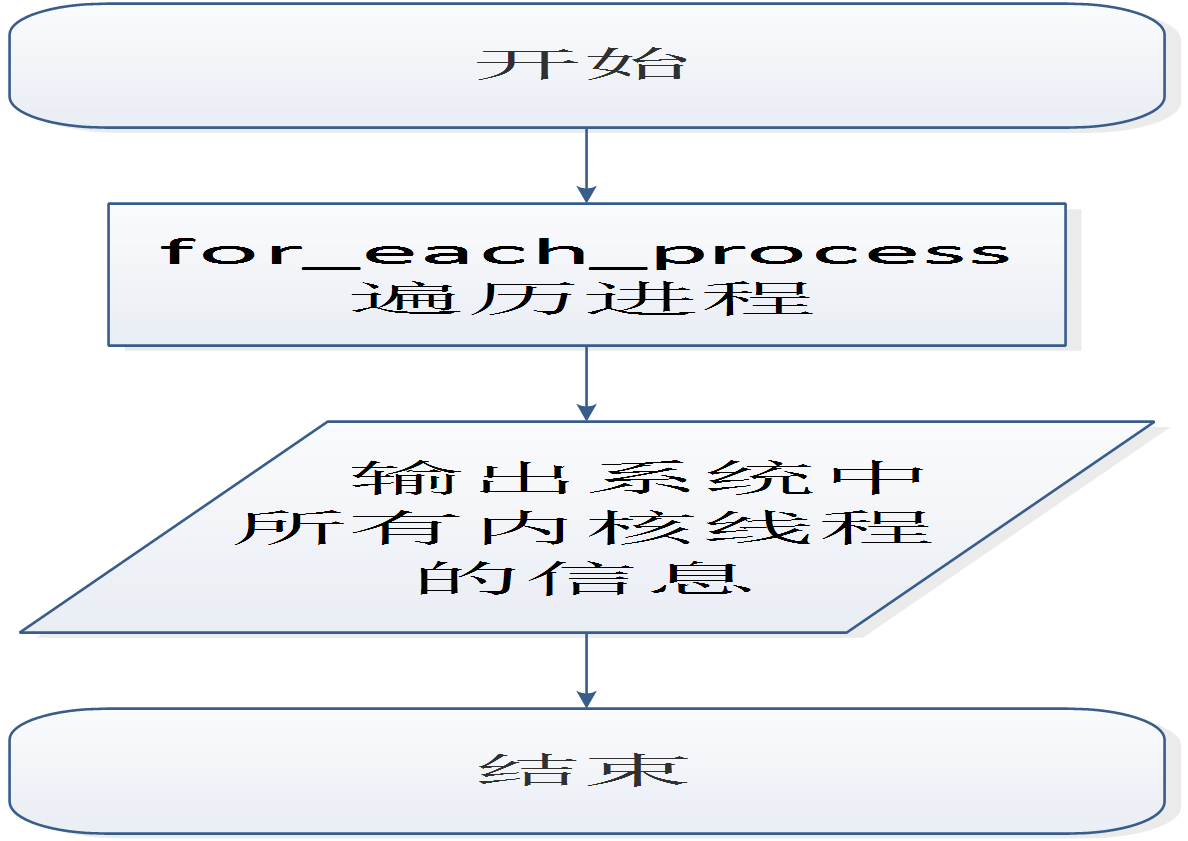
（2）本实验中主要函数的接口设计；

两个模块，xfmodule\_1和带参数为pid的xfmodule\_2

（3）主要函数的程序设计思路，并画出程序设计流程图。

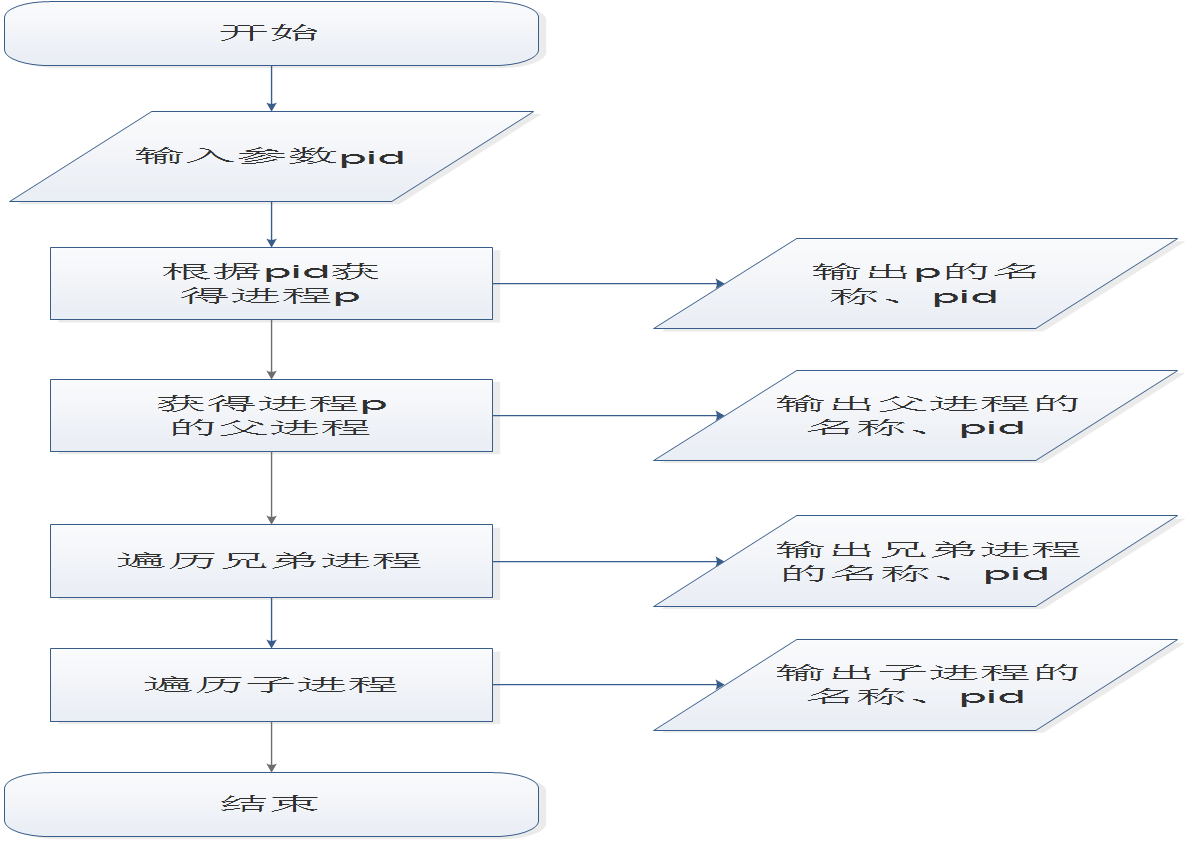
1. 模块1，xfmodule\_1.c

for\_each\_process遍历进程，输出系统中所有内核线程的信息



2. 模块2，xfmodule\_2.c

输入进程的pid，然后使用pid\_task函数根据pid找到进程，接着list\_for\_each遍历分别输出该进程的家族信息，包括父进程、兄弟进程和子进程的的程序名、PID号。



（4）对设计方案的创新性分析

模块二的时候，先输出了自己的信息，然后以列表的形式，先输出与参数进程的关系，再输出相关进程的信息

3、项目实现过程：

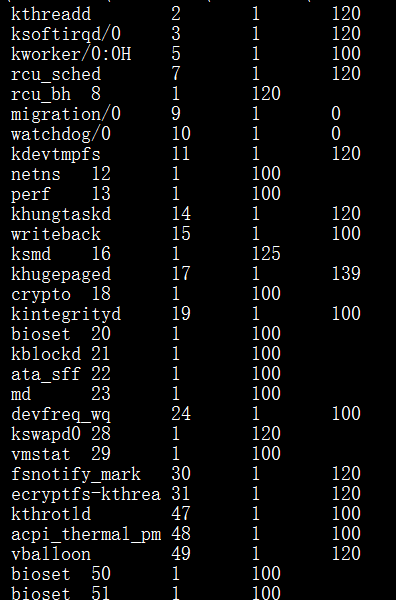
（1）详细记录项目实现过程中遇到的问题、原因及解决方法（可用截图加文字说明的方法进行编写）

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 问题 | 原因 | 解决方案 |
| 1 | 编译时候提示“warning: format '%d' expects argument of type 'int', but argument 4 has type 'long int' [-Wformat=]” | 返回格式问题 | 将“%d”改为“%ld” |
| 2 | insmod: ERROR: could not insert module xfmodule\_1.c: Invalid module format | 插入模块时  后缀输入错误 | Sudo insmod xfmodule\_1.ko |
| 3 | 模块插入是显示kill，之后再想插入即不显示东西，卡在那里 | 模块代码有问题，导致崩溃 | 给模块改个名字，然后再make&insmod或者重启电脑 |
| 4 | 子进程显示乱码及pid错误 | list\_entry函数参数错误 | 最后改为sibling，而不是task |

（2）分析程序运行结果

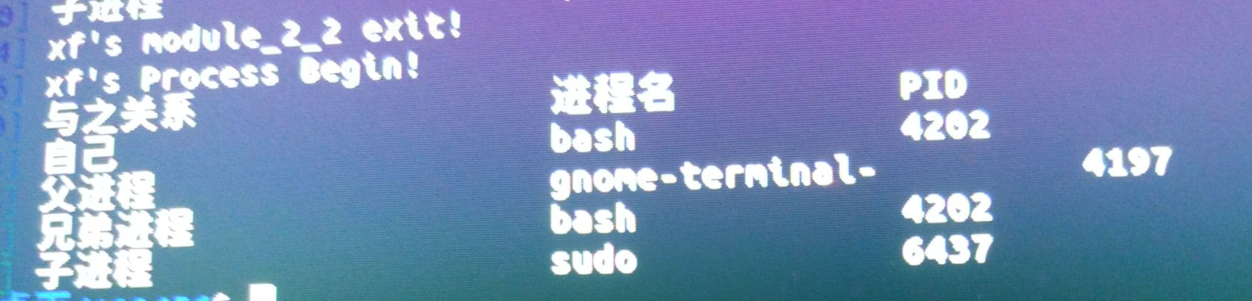
2.1 模块一

Insmod之后，显示所有进程的名称+pid+state+prio



2.2 模块二

加参数Insmod之后，显示改进程的名字+pid，然后再显示该进程父进程、兄弟进程、子进程的信息



（3）分析程序实现中的创新性；

模块二的时候，先输出了自己的信息，然后以列表的形式，先输出与参数进程的关系，再输出相关进程的信息

4、对项目的进一步思考：

（1）分析你所实现的程序的性能，包括优缺点；

模块简单，参考书籍做的，没有改进

（2）考虑的改进思路；

模块二的时候list\_for\_each之前的给list赋值，感觉不用，可以优化

（3）你认为教师对本实验的设计是否合理？难易程度如何？有什么改进建议？

合理，中等

5、参考文献（含阅读书籍、论文、网络资源等）

<http://blog.csdn.net/zhangskd/article/details/6856476>

<https://www.baidu.com/link?url=XScL2MN6QB2NS77jC_mFFIUFLXc6bI8pSBZTMrI0VbZGKOvmomnmnrGtoH242mD735Zf03YDHd32Sqelo5ZQ4a&wd=&eqid=8aefc5d000011b0f00000004592e7cfb>

6、程序完整代码

1. 模块一

**Xfmodule\_1.c**

/\*

\*@author：lxf

\*@time：2017/5/15 12：25

\*@title：os实验二第一题

\*@subject：设计一个模块，要求列出系统中所有内核线程

\* 的程序名、PID号、进程状态及进程优先级

\*/

#include<linux/init.h>

#include<linux/module.h>

#include<linux/kernel.h>

#include<linux/sched.h> //有进程描述符task\_struct

static int \_\_init xfmodule\_1\_init(void)

{

struct task\_struct \*p;

printk("Name\tPID\tState\tPrio\t\n");

for\_each\_process(p)

printk("%s\t%d\t%ld\t%d\n",p->comm,p->pid,p->state,p->prio);

return 0;

}

static void \_\_exit xfmodule\_1\_exit(void)

{

printk("xf's module01 exit!\n");

}

module\_init(xfmodule\_1\_init); //加载函数

module\_exit(xfmodule\_1\_exit); //卸载函数

MODULE\_LICENSE("GPL"); //许可证声明

**Makefile**

obj-m := xfmodule\_1.o

KDIR := /lib/modules/$(shell uname -r)/build

PWD := $(shell pwd)

default:

make -C $(KDIR) M=$(PWD) modules

clean:

make -C $(KDIR) M=$(PWD) clean

1. 模块二

**xfmodule\_2.c**

/\*

\*@author：lxf

\*@time：2017/5/15 14:39

\*@title：os实验二第二题

\*@subject：设计一个带参数的模块，其参数为某个进程的PID号，

\* 该模块的功能是列出改进程的家族信息，包括父进程，

\* 兄弟进程和子进程的的程序名、PID号

\*@ps:因为带参数，加载模块时要指出参数，如 sudo insmod xfmodule\_2.ko pid=xx

\*/

#include<linux/init.h>

#include<linux/module.h>

#include<linux/kernel.h>

#include<linux/sched.h> //有进程描述符task\_struct

#include<linux/moduleparam.h> //模块带参数

static int pid; //参数申明

module\_param(pid,int,0644); //参数说明

MODULE\_LICENSE("GPL");

static int \_\_init xfmodule\_2\_init(void)

{

struct task\_struct \*p;

struct task\_struct \*parent;

struct task\_struct \*process;

struct list\_head \*list;

printk("xf's Process Begin!\n");

printk("与之关系\t\t\t进程名\t\tPID\n");

//根据pid找到进程的地址

//p=find\_task\_by\_vpid(pid); 这个函数现在不管用啦

p=pid\_task(find\_vpid(pid),PIDTYPE\_PID);

printk("自己\t\t\t%s\t\t%d\n",p->comm,p->pid); //输出自己的信息

parent=p->parent; //父进程

printk("父进程\t\t\t%s\t\t%d\n",parent->comm,parent->pid);

list= &parent->children;

list\_for\_each(list,&parent->children) //遍历parent的children，即是他的sibling

{

process=list\_entry(list,struct task\_struct,sibling);

printk("兄弟进程\t\t\t%s\t\t%d\n",process->comm,process->pid);

}

list=&p->children; //子进程

list\_for\_each(list,&p->children) //遍历子进程

{

process=list\_entry(list,struct task\_struct,sibling);

printk("子进程\t\t\t%s\t\t%d\n",process->comm,process->pid);

}

return 0;

}

static void \_\_exit xfmodule\_2\_exit(void)

{

printk("xf's module\_2\_2 exit!\n");

}

module\_init(xfmodule\_2\_init);

module\_exit(xfmodule\_2\_exit);

**Makefile**

obj-m := xfmodule\_2.o

KDIR := /lib/modules/$(shell uname -r)/build

PWD := $(shell pwd)

default:

make -C $(KDIR) M=$(PWD) modules

clean:

make -C $(KDIR) M=$(PWD) clean