操作系统实验报告

实验二：Linux内核模块编译

XXXXXX

2019

目录

[LINUX内核模块编译 2](#_Toc11506546)

[一、 实验目的 2](#_Toc11506547)

[二、 实验要求 2](#_Toc11506548)

[三、 开发平台 2](#_Toc11506549)

[四、 实验设计方案 3](#_Toc11506550)

[五、 测试程序和测试截图 8](#_Toc11506551)

[六、 实验遇到的问题及解决方式 11](#_Toc11506552)

[七、 实验结果讨论 12](#_Toc11506553)

LINUX内核模块编译

# 实验目的

Linux提供的模块能动态扩充Linux功能而无需重新编译内核,以经广泛应用在Linux内核的许多功能的实现中。在本实验中将学习模块的基本概念、原理及实现技术，然后利用内核模块编程访问进程的基本信息，加深对进程的理解，掌握基本的模块编程技术。

# 实验要求

1. 设计一个模块，要求列出系统中所有内核线程的程序名、PID、进程状态、进程优先级、父进程的PID。
2. 设计一个带参数的模块，其参数为某个进程的PID号，模块的功能是列出该进程的家族信息，包括父进程、兄弟进程和子进程的程序名、PID号及进程状态。
3. 请根据自身情况，进一步阅读分析程序中用到的相关内核函数的源码实现。

# 开发平台

Ubuntu，gcc，vim，vscode，gdb等。

# 实验设计方案

模块一：列出系统中所有内核线程的程序名，pid，进程状态，进程优先级，父进程的pid。

模块设计

|  |
| --- |
|  |

模块代码如下

|  |
| --- |
| #include <linux/init.h>  #include <linux/module.h>  #include <linux/kernel.h>  #include <linux/sched/signal.h>  #include <linux/sched.h>  MODULE\_LICENSE("GPL");  **static** **int** \_\_init show\_all\_kernel\_thread\_init(**void**)  {  **struct** task\_struct \*p;  printk("%-20s%-6s%-6s%-6s%-6s", "Name", "PID", "State", "Prio", "PPID");  printk("--------------------------------------------");  for\_each\_process(p)  {  if (p->mm == NULL)  {  printk("%-20s%-6d%-6d%-6d%-6d", p->comm, p->pid, p->state, p->prio,  p->parent->pid);  }  }  return 0;  }  **static** **void** \_\_exit show\_all\_kernel\_thread\_exit(**void**)  {  printk("[显示所有内核线程]模块卸载");  }  module\_init(show\_all\_kernel\_thread\_init);  module\_exit(show\_all\_kernel\_thread\_exit); |

Makefile

|  |
| --- |
| obj-m := show\_all\_kernel\_thread.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 |

模块二：设计一个带参数的模块，参数为模块的pid，列出该进程的家族信息，包括父进程、兄弟进程和子进程的程序名、pid号及进程状态。

设计思路

|  |
| --- |
|  |

模块代码

|  |
| --- |
| #include <linux/init.h>  #include <linux/module.h>  #include <linux/kernel.h>  #include <linux/moduleparam.h>  #include <linux/pid.h>  #include <linux/list.h>  #include <linux/sched.h>  MODULE\_LICENSE("GPL");  **static** **int** pid;  module\_param(pid, **int**, 0644);  **static** **int** \_\_init show\_task\_family\_init(**void**)  {  **struct** pid \*ppid;  **struct** task\_struct \*p;  **struct** task\_struct \*pos;  **char** \*ptype[4] = {"[I]", "[P]", "[S]", "[C]"};  *// 通过进程的PID号pid一步步找到进程的进程控制块p*  ppid = find\_get\_pid(pid);  if (ppid == NULL)  {  printk("[ShowTaskFamily] Error, PID not exists.\n");  return -1;  }  p = pid\_task(ppid, PIDTYPE\_PID);  *// 格式化输出表头*  printk("%-10s%-20s%-6s%-6s\n", "Type", "Name", "PID", "State");  printk("------------------------------------------\n");  *// Itself*  *// 打印自身信息*  printk("%-10s%-20s%-6d%-6d\n", ptype[0], p->comm, p->pid, p->state);  *// Parent*  *// 打印父进程信息*  printk("%-10s%-20s%-6d%-6d\n", ptype[1], p->real\_parent->comm,  p->real\_parent->pid, p->real\_parent->state);  *// Siblings*  *// 遍历父进程的子，即我的兄弟进程，输出信息*  *// “我”同样是父进程的子进程，所以当二者进程PID号一致时，跳过不输出*  list\_for\_each\_entry(pos, &(p->real\_parent->children), sibling)  {  if (pos->pid == pid)  continue;  printk("%-10s%-20s%-6d%-6d\n", ptype[2], pos->comm, pos->pid,  pos->state);  }  *// Children*  *// 遍历”我“的子进程，输出信息*  list\_for\_each\_entry(pos, &(p->children), sibling)  {  printk("%-10s%-20s%-6d%-6d\n", ptype[3], pos->comm, pos->pid,  pos->state);  }  return 0;  }  **static** **void** \_\_exit show\_task\_family\_exit(**void**)  {  printk("[ShowTaskFamily] Module Uninstalled.\n");  }  module\_init(show\_task\_family\_init);  module\_exit(show\_task\_family\_exit); |

Makefile

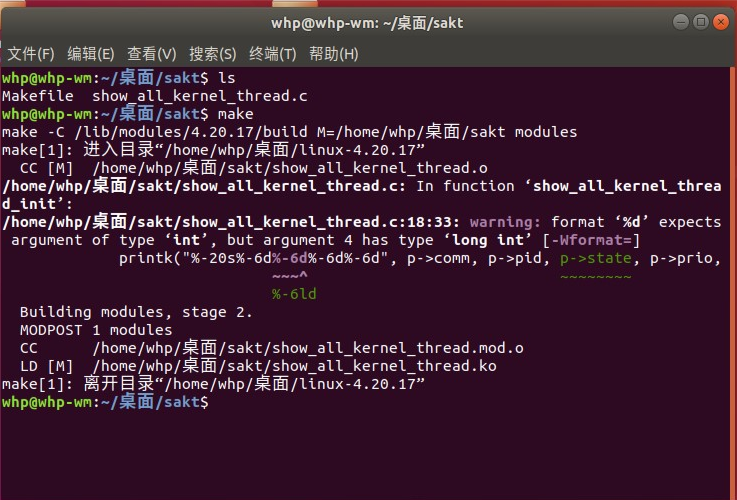
|  |
| --- |
| obj-m := show\_task\_family.o  KDIR := /lib/modules/$(shell uname -r)/build  PWD := $(shell pwd)  default:      make -C $(KDIR) M=$(PWD) modules  clean:      make -C $(KDIR) M=$(PWD) clean |

内核函数源码分析：

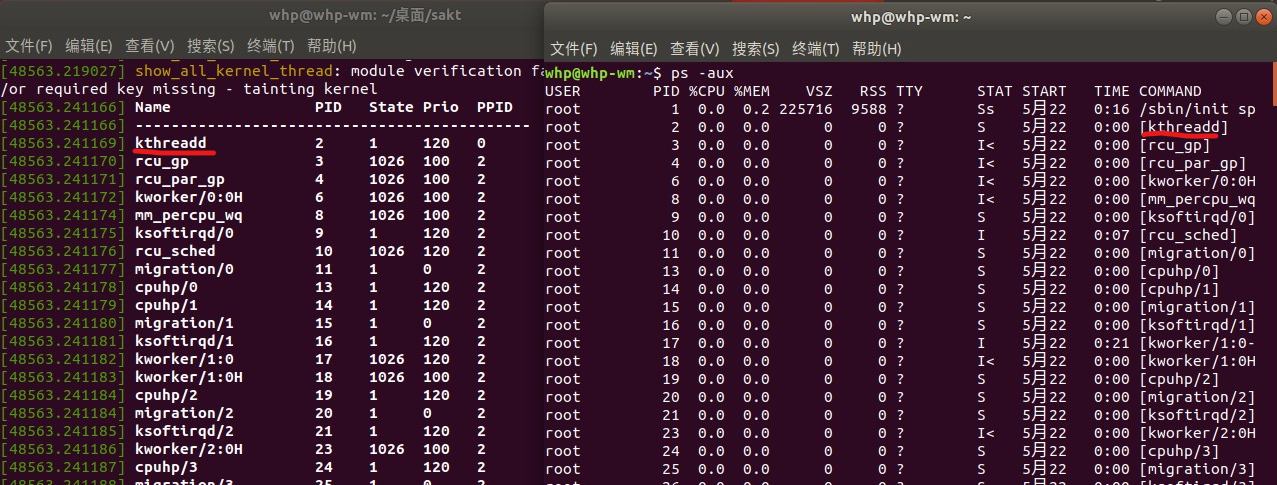
|  |
| --- |
| *// for\_each\_process*  *// 是一个宏定义：用来遍历每一个进程*  #define for\_each\_process(p) \      for (p = &init\_task ; (p = next\_task(p)) != &init\_task ; ) |
| *// task\_struct*  *// 是一个长达六百行的结构体里面包含进程的各种信息*  *// 本次用了，mm：判断是否为内核进程*  *// comm：程序名*  *// pid: 进程号*  *// state：进程状态*  *// prio：进程优先级*  *// parent：父进程的task\_struct* |
| printk()  内核函数：  用于将内容打印到内核缓冲区 |
| find\_get\_pid()  内核函数，用于获取pid结构体传入 参数为pid  返回一个pid结构体 |
| pid\_task()  内核函数，用于通过pid结构体找到进程控制块  传入参数为 pid结构体 和 类型  返回值为对应的 task\_struct |
| list\_for\_each\_entry()  宏定义：将head所指向的结构体指针，的member取出放到pos中  然后在下次循环中再次取出直到遍历结束  #define list\_for\_each\_entry(pos, head, member)              \      for (pos = list\_first\_entry(head, typeof(\*pos), member);    \       &pos->member != (head);                    \       pos = list\_next\_entry(pos, member)) |

# 测试程序和测试截图

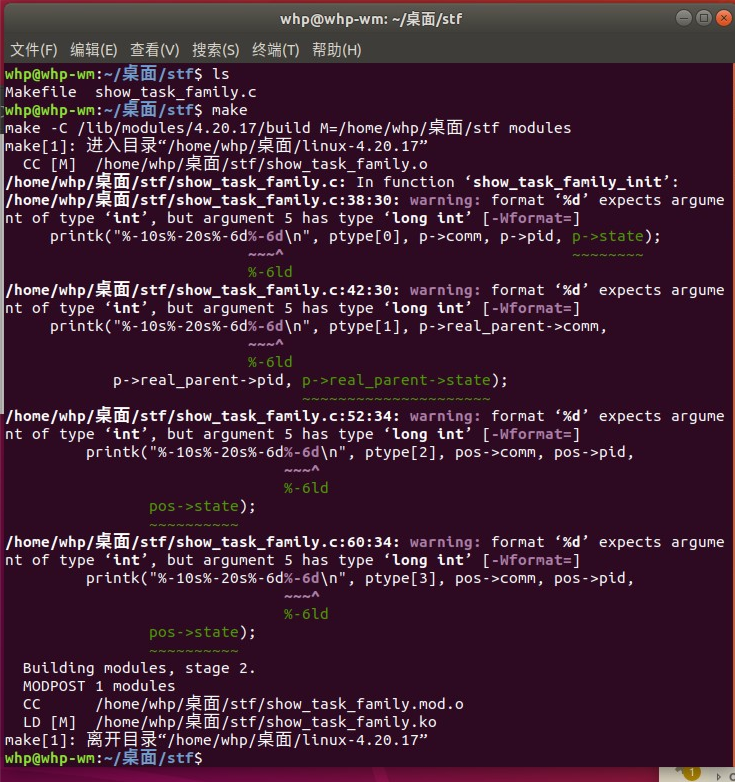
模块一编译



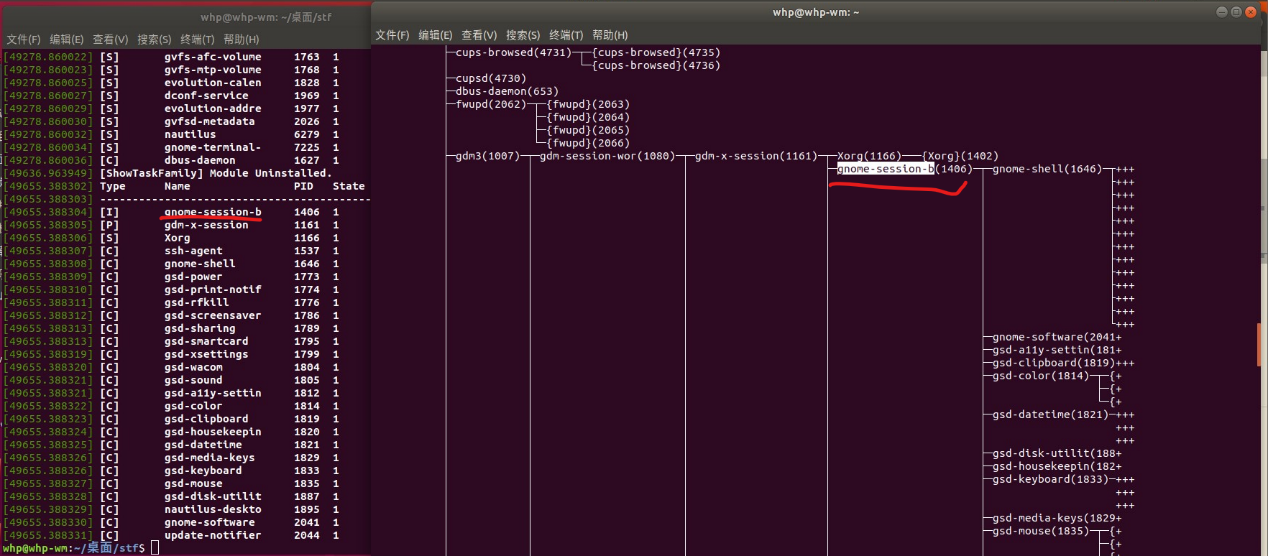
模块一显示



模块二编译



模块二显示



# 实验遇到的问题及解决方式

1. 模块编译的makefile的内核源码路径如何填？

/lib/modules/$(shell uname -r)/build

这个目录并没有所需要的源码啊

经过观察发现该目录的build文件是软连接到我们的内核源码目录的，所以直接按照书上的写法也没问题，经过实践得知直接填写绝对路径也没有问题

1. 为了测试父进程和子进程如何找到合适的进程pid来做测试

使用linux里面的pstree –p 就可以显示带pid的进程树,找到一个有父进程有兄弟进程又有子进程的用于测试即可。

1. 为了多测试下可以直接更改模块的参数吗？

不可，应该先将模块卸载再装入的时候再给定参数

如果重复装入模块就会报错提示file exists

1. 编译模块产生除.ko文件的其他文件这些是啥？如何清理？

是make中间文件，在目录下使用make clean清理不过所有生成的文件就会消失了。

# 实验结果讨论