

实验二 Linux 内核模块编程

设计要求
实验步骤
结果演示
体会

实验二 Linux 内核模块编程

17042127

陶逸群

设计要求

- 设计一个模块，要求列出系统中的所有内核线程的程序名、PID、进程状态、进程优先级、父进程PID。
- 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号以及进程状态。
- 模块二以树型方式打印输出
- 模块一按列对其输出
- 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。

实验步骤

1. 编写内核模块

- 打印出系统中所有内核线程信息的内核模块

```
1  #include <linux/init.h>
2  #include <linux/module.h>
3  #include <linux/kernel.h>
4  #include <linux/sched/signal.h>
5  // 模块在加载的时候会运行init函数
6  static int __init init_show_all_kernel_thread(void)
7  {
8      // 格式化输出头
9      struct task_struct *p;
10     printk("%-20s%-6s%-6s%-6s%-6s", "Name", "PID", "State", "Prio",
11     "PPID");
12     printk("-----");
13     // for_each_process(p)的作用是从0开始遍历进程链表中的所有进程
14     for_each_process(p)
15     {
16         // p最开始指向进程链表中第一个进程，随着循环不断进行p也不断后移直至链表
17         // 尾
18         if (p->mm == NULL){
19             // 打印进程p的相关信息
20             printk("%-20s%-6d%-6d%-6d%-6d", p->comm, p->pid, p->state, p->prio,
21             p->parent->pid);
22         }
23     }
24     return 0;
25 }
26 // 模块在加载的时候会运行exit函数
27 static void __exit exit_show_all_kernel_thread(void)
28 {
```

```

29     printk("-----所有进程显示完毕-----\n");
30 }
31 module_init(init_show_all_kernel_thread);
32 module_exit(exit_show_all_kernel_thread);
33
34 MODULE_LICENSE("GPL");

```

内核函数的具体作用见以上注释部分。

- 以树形打印某进程家族信息的内核模块

```

1  #include <linux/init.h>
2  #include <linux/module.h>
3  #include <linux/kernel.h>
4  #include <linux/moduleparam.h>
5  #include <linux/pid.h>
6  #include <linux/list.h>
7  #include <linux/sched.h>
8
9  static int pid;
10 module_param(pid, int, 0644);
11
12 static int __init show_task_family_init(void){
13     struct pid *ppid;
14     struct task_struct *p;
15     struct task_struct *pos;
16     struct list_head *pos_head;
17     int i=0;
18     // 通过进程的PID号pid一步步找到进程的进程控制块p
19     ppid = find_get_pid(pid);
20     if (ppid == NULL){
21         printk("pid不存在\n");
22         return -1;
23     }
24     p = pid_task(ppid, PIDTYPE_PID);
25     list_for_each(pos_head,&(p->children)){
26         pos = list_entry(pos_head,struct task_struct,sibling);
27         if(i==0){
28             printk("%-10s(%-4d){%-6ld}---%-10s(%-4d){%-6ld}---%-10s(%-4d)
29             {%-6ld}", p->real_parent->comm,
30             p->real_parent->pid,p->real_parent->state, p->comm, p-
31             >pid, p->state,pos->comm, pos->pid,pos->state);
32         }else{
33             printk("%*s|%-10s(%-4d){%-6ld}",25,"",26,"", pos->comm,
34             pos->pid,pos->state);
35         }
36         i++;
37     }
38     if(i==0){
39         printk("%-10s(%-4d){%-6ld}---%-10s(%-4d){%-6ld}", p-
40         >real_parent->comm,
41         p->real_parent->pid,p->real_parent->state, p->comm, p-
42         >pid, p->state);
43     }
44     list_for_each(pos_head,&(p->sibling)){
45         if(pos_head!= &p->parent->children){
46             pos = list_entry(pos_head,struct task_struct,sibling);
47             printk("%*s|%-10s(%-4d){%-6ld}",25,"", pos->comm, pos-
48             >pid,pos->state);
49         }
50     }
51     return 0;
52 }

```

```

48
49 static void __exit show_task_family_exit(void)
50 {
51     printk("模块运行完毕\n");
52 }
53 MODULE_LICENSE("GPL");
54 module_init(show_task_family_init);
55 module_exit(show_task_family_exit);

```

在该模块中，首先调用函数 `find_get_pid()` 并传入外部参数——某进程的 `pid` 号来获得该进程的进程描述符，并调用函数 `pid_task()` 获得该进程的进程描述符信息，然后通过使用 Linux 内核中定义的 `list_for_each()` 宏和 `list_entry()` 宏分别遍历该进程的子进程链表和兄弟进程链表，控制输出为树型。

2. 编写模块编译的 Makefile 文件

- 打印出系统中所有内核线程信息的内核模块的 Makefile 文件

```

1 obj-m := show_all_kernel_thread_states.o
2 KDIR := /lib/modules/$(shell uname -r)/build
3 # 当前路径
4 PWD := $(shell pwd)
5 default:
6     make -C $(KDIR) M=$(PWD) modules
7 clean:
8     make -C $(KDIR) M=$(PWD) clean

```

- 以树形打印某进程家族信息的内核模块的 Makefile 文件

```

1 obj-m := show_task_family.o
2 KDIR := /lib/modules/$(shell uname -r)/build
3 # 当前路径
4 PWD := $(shell pwd)
5 default:
6     make -C $(KDIR) M=$(PWD) modules
7 clean:
8     make -C $(KDIR) M=$(PWD) clean

```

3. 编译并加载内核模块

结果演示

- 打印出系统中所有内核线程信息

root@localhost module_2# ps - aux										
USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.1	0.2	193808	8536	?	Ss	19:47	0:04	/usr/lib/syste
root	2	0.0	0.0	0	0	?	S	19:47	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	19:47	0:00	[rcu_gp]
root	4	0.0	0.0	0	0	?	I<	19:47	0:00	[rcu_par_gp]
root	6	0.0	0.0	0	0	?	I<	19:47	0:00	[kworker/0:0H-
root	8	0.0	0.0	0	0	?	I<	19:47	0:00	[mm_percpu_wq]
root	9	0.0	0.0	0	0	?	S	19:47	0:00	[ksoftirqd/0]
root	10	0.0	0.0	0	0	?	I	19:47	0:03	[rcu_sched]
root	11	0.0	0.0	0	0	?	S	19:47	0:00	[migration/0]
root	13	0.0	0.0	0	0	?	S	19:47	0:00	[cpuhp/0]
root	14	0.0	0.0	0	0	?	S	19:47	0:00	[cpuhp/1]
root	15	0.0	0.0	0	0	?	S	19:47	0:00	[migration/1]
root	16	0.0	0.0	0	0	?	S	19:47	0:00	[ksoftirqd/1]
root	18	0.0	0.0	0	0	?	I<	19:47	0:00	[kworker/1:0H-
root	19	0.0	0.0	0	0	?	S	19:47	0:00	[cpuhp/2]
root	20	0.0	0.0	0	0	?	S	19:47	0:00	[migration/2]
root	21	0.0	0.0	0	0	?	S	19:47	0:00	[ksoftirqd/2]
root	23	0.0	0.0	0	0	?	I<	19:47	0:00	[kworker/2:0H-
root	24	0.0	0.0	0	0	?	S	19:47	0:00	[cpuhp/3]
root	25	0.0	0.0	0	0	?	S	19:47	0:00	[migration/3]
root	26	0.0	0.0	0	0	?	S	19:47	0:00	[ksoftirqd/3]
root	28	0.0	0.0	0	0	?	I<	19:47	0:00	[kworker/3:0H-

与 ps - aux 输出一致。

root@localhost module_2# ps - aux										
USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.1	0.2	193808	8536	?	Ss	19:47	0:04	/usr/lib/syste
root	2	0.0	0.0	0	0	?	S	19:47	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	19:47	0:00	[rcu_gp]
root	4	0.0	0.0	0	0	?	I<	19:47	0:00	[rcu_par_gp]
root	6	0.0	0.0	0	0	?	I<	19:47	0:00	[kworker/0:0H-
root	8	0.0	0.0	0	0	?	I<	19:47	0:00	[mm_percpu_wq]
root	9	0.0	0.0	0	0	?	S	19:47	0:00	[ksoftirqd/0]
root	10	0.0	0.0	0	0	?	I	19:47	0:03	[rcu_sched]
root	11	0.0	0.0	0	0	?	S	19:47	0:00	[migration/0]
root	13	0.0	0.0	0	0	?	S	19:47	0:00	[cpuhp/0]
root	14	0.0	0.0	0	0	?	S	19:47	0:00	[cpuhp/1]
root	15	0.0	0.0	0	0	?	S	19:47	0:00	[migration/1]
root	16	0.0	0.0	0	0	?	S	19:47	0:00	[ksoftirqd/1]
root	18	0.0	0.0	0	0	?	I<	19:47	0:00	[kworker/1:0H-
root	19	0.0	0.0	0	0	?	S	19:47	0:00	[cpuhp/2]
root	20	0.0	0.0	0	0	?	S	19:47	0:00	[migration/2]
root	21	0.0	0.0	0	0	?	S	19:47	0:00	[ksoftirqd/2]
root	23	0.0	0.0	0	0	?	I<	19:47	0:00	[kworker/2:0H-
root	24	0.0	0.0	0	0	?	S	19:47	0:00	[cpuhp/3]
root	25	0.0	0.0	0	0	?	S	19:47	0:00	[migration/3]
root	26	0.0	0.0	0	0	?	S	19:47	0:00	[ksoftirqd/3]
root	28	0.0	0.0	0	0	?	I<	19:47	0:00	[kworker/3:0H-

- 以树形打印某进程家族信息

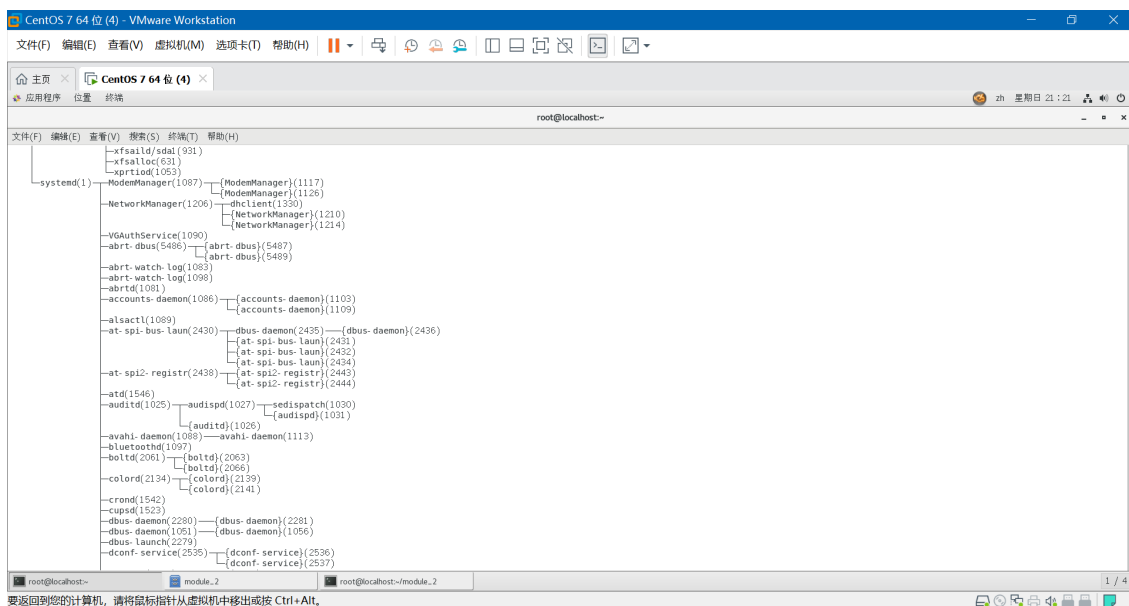
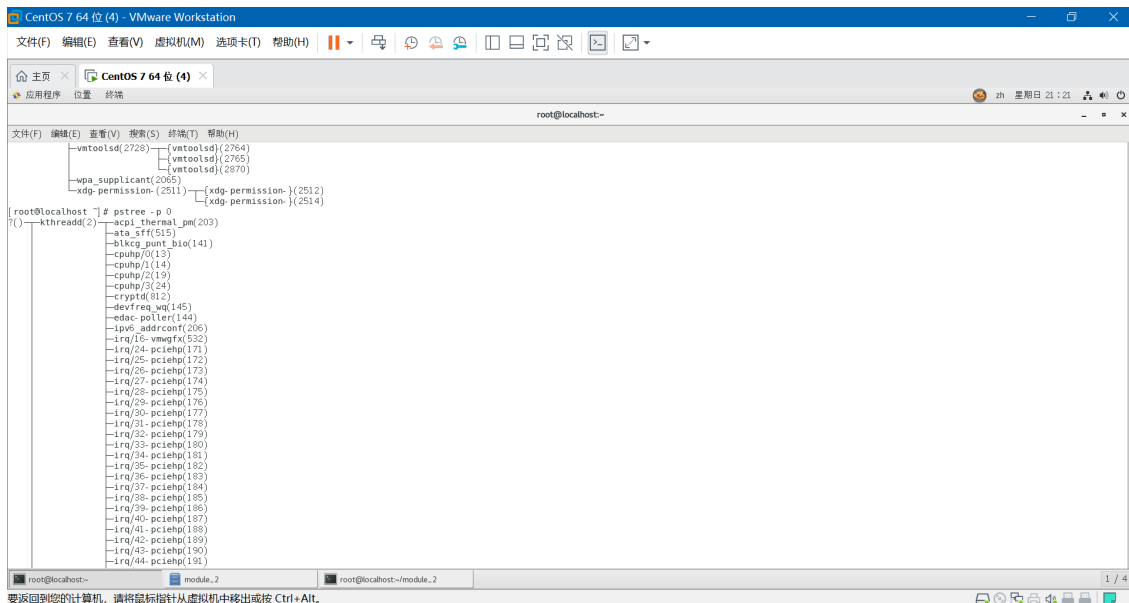
3854.603761	swapper/0 (0) (0)	...	systemd (1) (1)	...	systemd-journal(723) (1)	
3854.603762					-lvmtd (749) (0)	
3854.603763					-systemd-udevd(760) (0)	
3854.603764					-auditd (1026) (1)	
3854.603765					-polkitd (1050) (1)	
3854.603766					-dbus-daemon(1051) (1)	
3854.603767					-rngd (1061) (1)	
3854.603767					-rtkit-daemon(1064) (1)	
3854.603768					-systemd-logind(1071) (1)	
3854.603769					-smartd (1074) (1)	
3854.603770					-udisksd (1075) (1)	
3854.603771					-lsmd (1076) (1)	
3854.603771					-irqbalance(1080) (1)	
3854.603772					-abrt (1081) (1)	
3854.603774					-abrt-watch-log(1083) (1)	
3854.603775					-accounts-daemon(1086) (1)	
3854.603776					-ModemManager(1087) (1)	
3854.603776					-avahi-daemon(1088) (1)	
3854.603779					-alsactl (1089) (1)	
3854.603799					-VidurService(1090) (1)	
3854.603800					-vmttoolsd (1096) (1)	
3854.603801					-bluetoothd(1097) (1)	
3854.603801					-abrt-watch-log(1098) (1)	
3854.603802					-rpcbind (1118) (1)	
3854.603803					-gssproxy (1112) (1)	
3854.603804					-firewalld(1127) (1)	
3854.603805					-ksmtuned (1142) (1)	
3854.603806					-NetworkManager(1206) (1)	
3854.603807					-sshd (1520) (1)	
3854.603807					-tuned (1522) (1)	
3854.603808					-cupsd (1523) (1)	
3854.603808					-rsyslogd (1527) (1)	
3854.603810					-libvirtd (1531) (0)	

```
3854,603821 }
3854,603821 }
3854,603822 }
3854,603823 }
3854,603824 }
3854,603825 }
3854,603826 }
3854,603827 }
3854,603828 }
3854,603829 }
3854,603830 }
3854,603831 }
3854,603832 }
3854,603833 }
3854,603833 }
3854,603834 }
3854,603835 }
3854,603836 }
3854,603837 }
3854,603838 }
3854,603839 }
3854,603839 }
3854,603840 }
3854,603841 }
3854,603842 }
3854,603843 }
3854,603844 }
3854,603845 }
3854,603846 }
3854,603847 }
3854,603848 }

-ibus-daemon(2252){1 }
-ibus-daemon(2280){1 }
-ibus-launch(2279){1 }
-gvfsd(2315){1 }
-gvfsd-fuse(2328){1 }
-ismettings-daemon(2311){1 }
-at-spi-bus-launch(2430){1 }
-at-spi2-registry(2438){1 }
-ibus-x11(2498){1 }
-ibus-portal(2500){1 }
-xdg-permission(2511){1 }
-evolution-source(2521){1 }
-geoclue(2530){1 }
-gnome-shell-call(2516){1 }
-dconf-service(2535){1 }
-goa-identity-se(2544){1 }
-gvfs-udisks2-vol(2557){1 }
-gvfs-mtp-volume(2562){1 }
-goa-daemon(2527){1 }
-gvfs-afc-volume(2567){1 }
-mission-control(2551){1 }
-gvfs-goa-volume(2574){1 }
-gvfs-gphoto2-vol(2579){1 }
-gssd-printer(2665){1 }
-pulseaudio(2626){1 }
-evolution-calendar(2713){1 }
-vntoolsd(2728){1 }
-tracker-store(2775){1 }
-evolution-address(2753){1 }
-nautilus(2830){1 }
-fwupd(2879){1 }
-gvfs-metadata(2085){1 }
-gnome-terminal(4525){1 }

-kthreadd(2){1 }
```

这里的pid为1, 与 `ps tree - p 0` 结果一致



体会

本次实验让我对 Linux 系统以及 Linux 内核有了更加深入的认识, 编写内核模块可以实现对内核功能的动态添加, 与通过直接修改内核添加系统调用实现添加内核功能相比更加灵活。同时, 通过本次实验我还了解了一些 Linux 内核中定义的宏, 比如 `list_for_each()`、`for_each_process()`、以及 `list_entry()`。