# 操作系统课程设计报告

**实验二：Linux 内核模块编程**

**专业： 理工类实验班**

**班级： 19185312**

**姓名： 郑凯心**

**学号： 19063140**

**指导老师： 任彧**

**上机时间：周五下午6、7、8**

**杭州电子科技大学 卓越学院**

**2021.10**

## 一、实验目的

Linux 提供的模块机制能动态扩充 linux 功能而无需重新编译内核，已经广泛应用在 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，加深对进程概念的理解，掌握基本的模块编程技术。

### 二、实验内容

（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、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。

### 三、实验方法

**模块编译和加载**

① 模块编译命令 make：

命令格式：make

不带参数的make命令将默认当前目录下名为makefile或者Makefile的文件为描述文件。

② 加载模块命令 insmod 或 modprobe：

insmod 命令把需要载入的模块以目标代码的形式加载到内核中，将自动调用

init\_module 宏。其格式为：

insmod [filename] [module options...]

modprobe 命令的功能与 insmod 一样，区别在于 modprobe 能够处理 module 载入的

相依问题，其格式为：

modprobe [module options...] [modulename] [module parameters...]

如本示例中加载模块的命令为：insmod hello.ko

③ 查看已加载模块命令 lsmod：

列出当前所有已载入系统的模块信息，包括模块名、大小、其他模块的引用计数等信息。

命令格式： lsmod

可以配合 grep 来查看指定模块是否已经加载：lsmod | grep 模块名

④ 查看指定模块信息命令 modinfo

查看指定模块的详细信息，如模块名、作者、许可证、参数等信息。

命令格式： modinfo 模块名

⑤卸载模块命令 rmmod：

卸载已经载入内核的指定模块，命令格式为：rmmod 模块名

**Linux 内核链表结构及操作**

链表是 linux 内核中最简单、最常用的一种数据结构。Linux 内核对链表的实现方式与众

不同，它给出了一种抽象链表定义，实际使用中可将其嵌入到其他数据结构中，从而演化出

所需要的复杂数据结构。

（1）链表的定义：

Linux 中链表的定义为：

struct list\_head {

struct list\_head \*next, \*prev;

}

这个不含数据域的链表，可以嵌入到任何结构中，形成结构复杂的链表，之后就以

struct list\_head 为基本对象，进行链表的插入、删除、合并、遍历等各种操作。如：

struct numlist {

int num;

struct list\_head list;

};

（2）链表的操作

① list\_for\_each()宏和 list\_entry()宏：

Linux 内核为抽象链表定义了若干操作，如申明及初始化链表、插入节点、删除节点、

合并链表、遍历链表等。本实验只涉及读取内核已有链表，所以这里只介绍链表遍历操作，

感兴趣的读者可以查看 linux-4.12/include/linux/list.h 文件学习。

list.h 中定义了遍历链表的宏：

#define list\_for\_each(pos, head) \

for (pos = (head)->next; pos != (head); pos = pos->next)

这个宏仅仅是找到一个个节点中 list\_head 域的位置 pos。

② list\_for\_each\_entry()宏

#define list\_for\_each\_entry(pos, head, member) \

for (pos = list\_entry((head)->next, typeof(\*pos), member); \

prefetch(pos->member.next), &pos->member != (head); \

pos = list\_entry(pos->member.next, typeof(\*pos), member))

该宏实际上是一个 for 循环，利用 pos 作为循环变量，从传入的链表头 head 开始，

逐项向后（next 方向）移动 pos，直至又回到 head。prefetch() 可以不考虑，用于预取以提高遍历速度。

2. 进程的 task\_struct 结构及家族关系

Linux 进程描述符 task\_struct 结构定义在 linux-4.12/include/linux/sched.h 中，包含众多的成员项，其中与本实验相关的成员项有：state、 prio、mm、parent、children、sibling、comm 等，请读者自行查看相关源码及本教材“3.3.4 Linux 进程管理”一节内容。

实验内容（1）可以利用内核的进程总链表实现，每个进程通过 task\_struct 结构的

next\_task, prev\_task 成员加入该链表。Linux 内核提供了宏 for\_each\_process()依次访问该链表中的每个进程，定义在/include/linux/sched.h 文件中：

#define for\_each\_process(p) \

for (p = &init\_task ; (p = next\_task(p)) != &init\_task ; )

实验内容（2）需要了解 linux 进程家族的组织情况，由 task\_struct 中的 parent、children、sibling 三个成员来描述，相关内容可参见前面“3.3.4Linux 进程管理”。

对子进程链表和兄弟进程链表的访问，可通过宏 list\_for\_each()和 list\_entry()以及

list\_for\_each\_entry()来实现。对于指定的 pid，可通过函数 pid\_task()和 find\_vpid()（或者find\_get\_pid()）配合使用找到其相应的 task\_atruct 结构，相关函数定义在 linux/kernel/pid.c文件中。

在第一个任务列出所以内核线程中，先枚举所有进程，进程结构task\_struct中mm成员指向进程所拥有的内存描述符，内核线程不拥有任何内存描述符，所以内核线程的mm为NULL。格式化输出即可。

1. for\_each\_process(p) {
2. **if** (p->mm == NULL) {
3. printk(KERN\_ALERT"%20s %6d %6ld %6d %6d\n",\
4. p->comm, p->pid, p->state, p->prio, p->parent->pid);
5. }
6. }

在第二个任务中，首先判断输入是否合法即进程是否存在，然后先输出父亲进程，再输出兄弟进程以及子进程。

各函数功能介绍

1. **int** print\_task(**struct** task\_struct \*p);

输入一个进程，格式化输出进程信息，提取出该功能目的是为了简化代码。

1. **int** count\_children(**struct** task\_struct \*p);

传入一个进程返回该进程子进程的数量，提取出该功能目的是为了简化代码。

1. **void** show\_thread(**struct** task\_struct \*p, **int** parent\_len, **int** lastflag);

功能是打印进程及其子进程,传入进程，以及其父亲的长度parent\_len为了方便输出定位，lastflag是为了表示该进程是否为其父进程的最后一个进程，如果是则需要特殊的符号。

打印过程中需要判断各种条件来保证格式正确

1. 是否有父亲进程
2. 兄弟进程数量，兄弟进程的首个与最后一个和中间的格式个不相同
3. 子进程数量，子进程的首个与最后一个和中间的格式个不相同

为了避免程序变得复杂使用过多的 if 判断语句，代码使用条件判断加数组下标来实现不同目录格式的输出。字符数组定义如下

1. **char** tree\_symbol[4][10] = {"├─", "└─", "─┬─", "───"};

- 如果只有一个则选择tree\_symbol[3]

- 如果是第一个则选择tree\_symbol[2]

- 如果是中间则选择tree\_symbol[0]

- 如果是最后一个则选择tree\_symbol[1]

例如以下函数功能为打印进程及其子进程

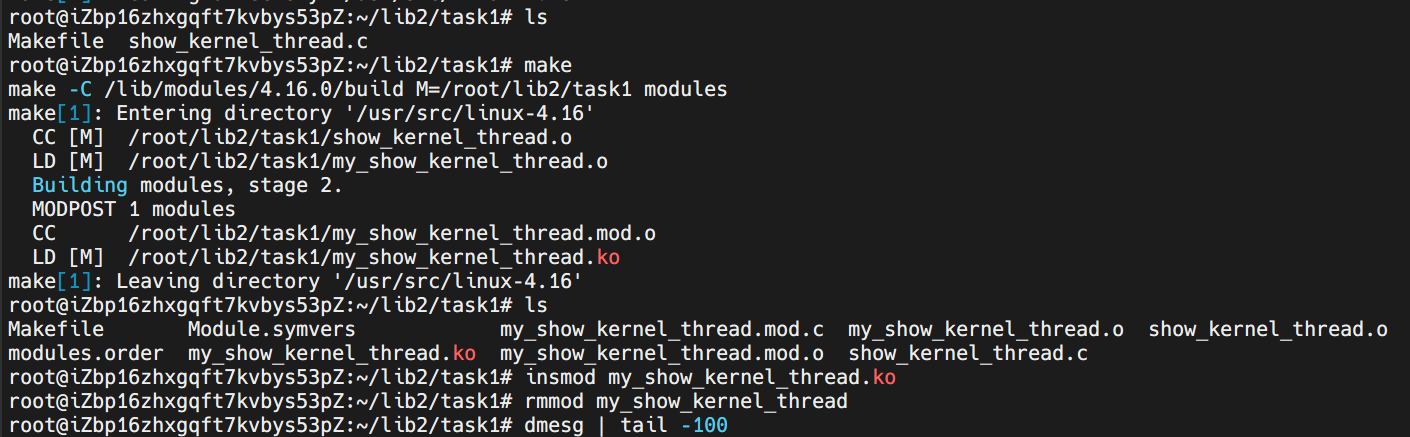
1. **void** show\_thread(**struct** task\_struct \*p, **int** parent\_len, **int** lastflag) {
2. **struct** list\_head \*list\_ptr;
3. **struct** task\_struct \*pchild;
4. **int** num\_children = count\_children(p), cur = 0;
5. **int** p\_len = print\_task(p)+2;
6. **if** (num\_children == 0) {
7. printk(KERN\_CONT"\n");
8. **return**;
9. }
10. list\_for\_each(list\_ptr, &p->children) {
11. pchild = list\_entry(list\_ptr, **struct** task\_struct, sibling);
12. **if** (++cur > 1) printk(KERN\_CONT"%\*s%s%\*s", parent\_len, "", lastflag ? " " : "│", p\_len, "");
13. printk(KERN\_CONT"%s", tree\_symbol[cur == 1 ? 2+(num\_children == 1) : cur == num\_children]);
14. print\_task(pchild);
15. printk(KERN\_CONT"\n");
16. }
17. }

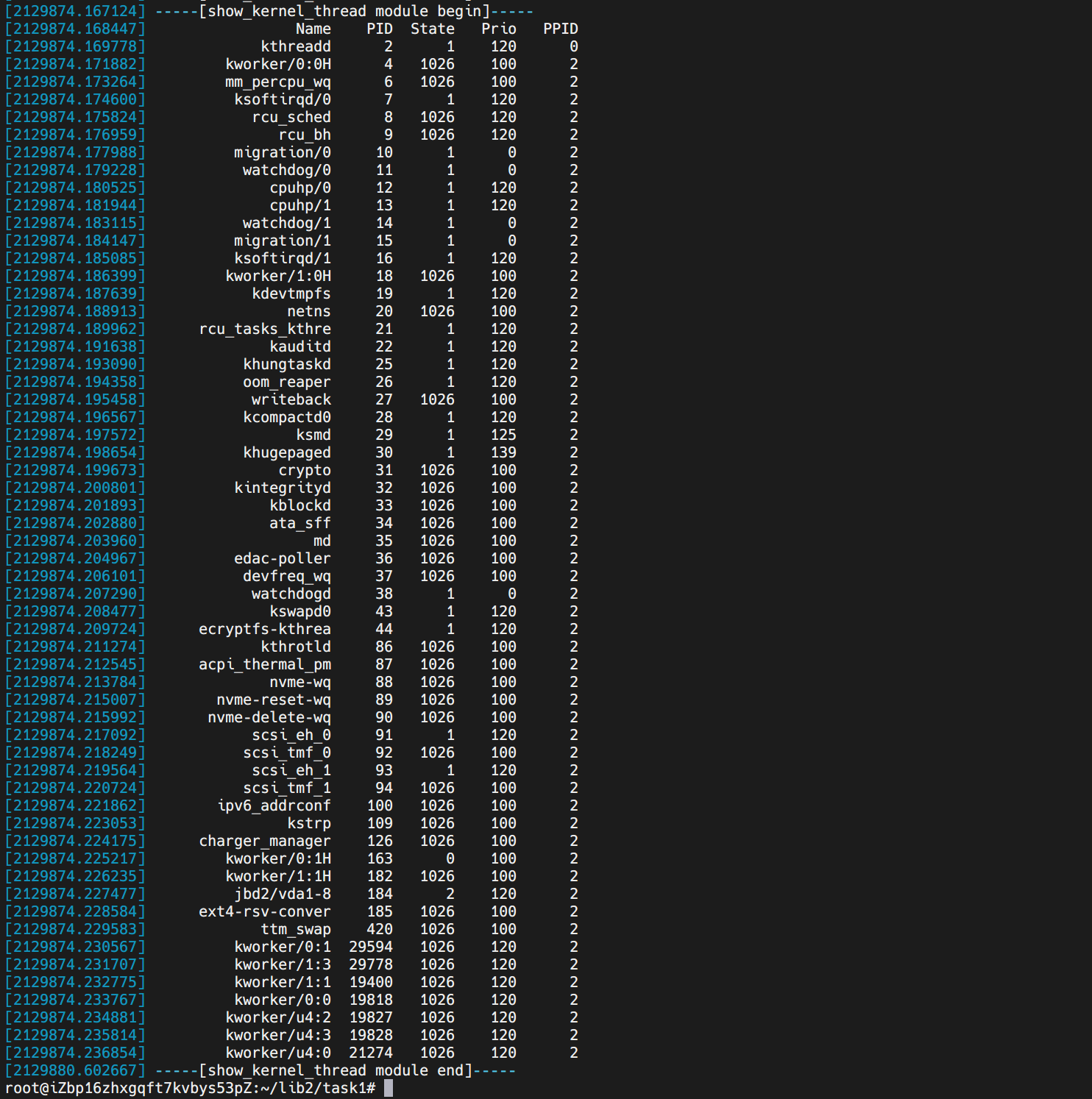
13行中cur为当前子进程是第几个，num\_children是子进程的个数，通过表达式 cur == 1 ? 2+(num\_children == 1) : cur == num\_children] 即可计算出相应的格式。

更多详情见附录代码。

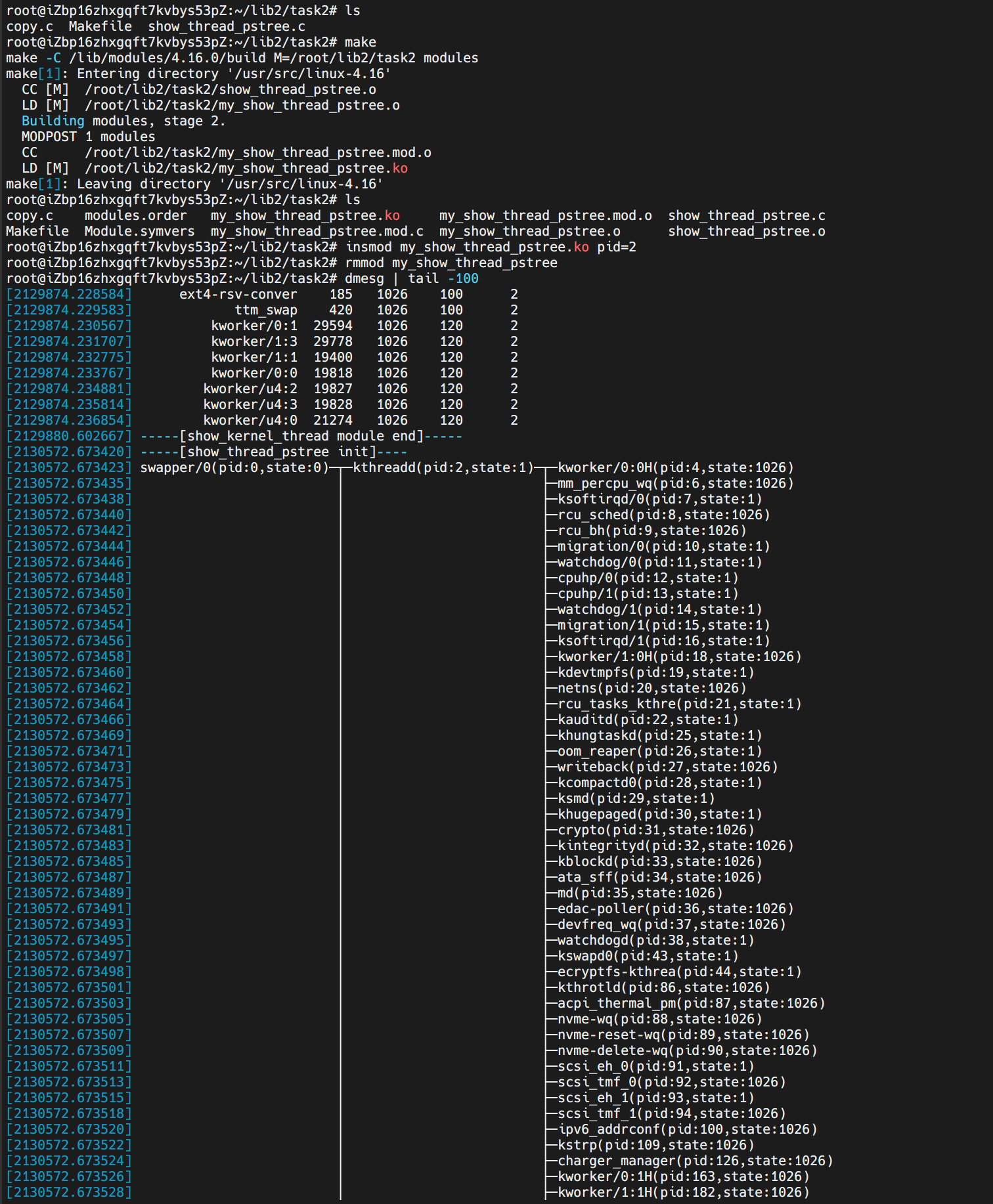
### 四、实验过程和结果

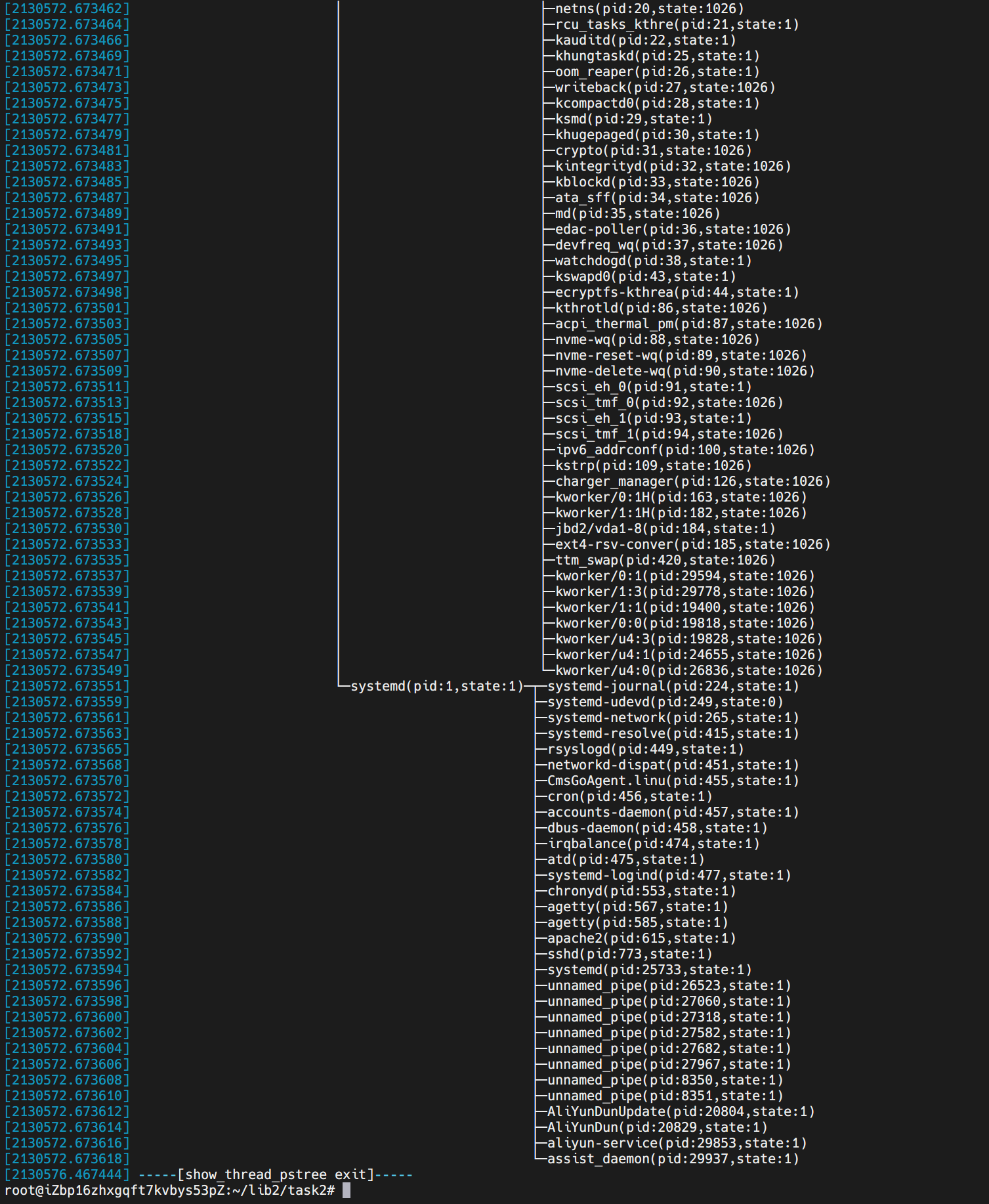
列出系统中所有内核线程的程序名、PID、进程状态、进程优先级、父进程的 PID。





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、进程状态。





### 五、实验体会

遇到的问题和分析：第二个写完后运行发现程序崩溃，后排查发现没有特判传入的进程是否存在。

实验总结：linux 提供了内核模块（module）机制，它不仅可以弥补单体式内核相对于微内核的一些不足，而且不影响系统性能。通过本次实验，我对：linux内核模块有了初步的了解，学会了如何编写编译并装载模块，还对linux内核链表结构与操作有了初步了解，在编写程序的过程中，实际上并不需要很深刻的思维，需要考虑的是很多细节，像如何判断进程的位置和数量对此决定该输出怎样的目录表使之看起来合理，如果单单使用简单的if语句来逐一判断，不仅会让代码变得冗长难以阅读难以维护，而且复杂之后很可能漏掉一些情况，这激发了我的思考做出了优化，通过一些简单的逻辑计算和下标定位来确定一个进程应该选择怎样目录符号，通过这个任务的编程让我受益匪浅。

### 六、参考文献

计算机操作系统(杭电编)