专题7-Linux内核链表

一、链表对比

1.1、链表简介

链表是一种常用的数据结构，它通过指针将一系列数据节点连接成一条数据链。相对于数组，链表具有更好的动态性，建立链表时无需预先知道数据总量，可以随机分配空间，可以高效地在链表中的任意位置实时插入或删除数据。链表的开销主要是访问的顺序性和组织链的空间损失。

1.2、内核链表结构

```
struct list_head
{
struct list_head *next, *prev;
};
```

list_head结构包含两个指向list_head结构的指针 prev和next，由此可见，内核的链表具备双链表功能，实际上，通常它都组织成双向循环链表。

1.3、内核链表-函数

1. INIT_LIST_HEAD:创建链表
2. list_add：在链表头插入节点
3. list_add_tail：在链表尾插入节点
4. list_del ：删除节点
5. list_entry：取出节点
6. list_for_each：遍历链表

二、内核链表使用

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/list.h>

struct score
{
 int num;
 int english;
 int math;
 struct list_head list;
};

struct list_head score_head;
struct score stu1, stu2, stu3;
struct list_head * pos;
struct score *tmp;

int mylist_init(void)
{
 INIT_LIST_HEAD(&score_head);

 stu1.num = 1;
 stu1.english = 90;
 stu1.math = 98;
 list_add_tail(&(stu1.list), &(score_head));

 stu2.num = 2;
 stu2.english = 86;
 stu2.math = 75;
 list_add_tail(&(stu2.list), &(score_head));

 stu3.num = 3;
 stu3.english = 84;
 stu3.math = 97;
 list_add_tail(&(stu3.list), &(score_head));

 list_for_each(pos, &(score_head))
 {
  tmp = list_entry(pos, struct score, list);
```

```
  printk(KERN_WARNING"No %d, english is %d, math is %d\n\r", tmp->num, tmp->english, tmp->math);
 }

 return 0;
}

void mylist_exit(void)
{
 list_del(&(stu1.list));
 list_del(&(stu2.list));
}

module_init(mylist_init);
module_exit(mylist_exit);

MODULE_AUTHOR("JOHNSON");
MODULE_LICENSE("Dual BSD/GPL");
MODULE_DESCRIPTION("linux kernel link list");
```

按照这代码安装模块后却出现这样的结果，原因未知，后续再做研究：

三、内核链表实现分析

四、移植内核链表