1.3 Linux内核源码入门-双链表

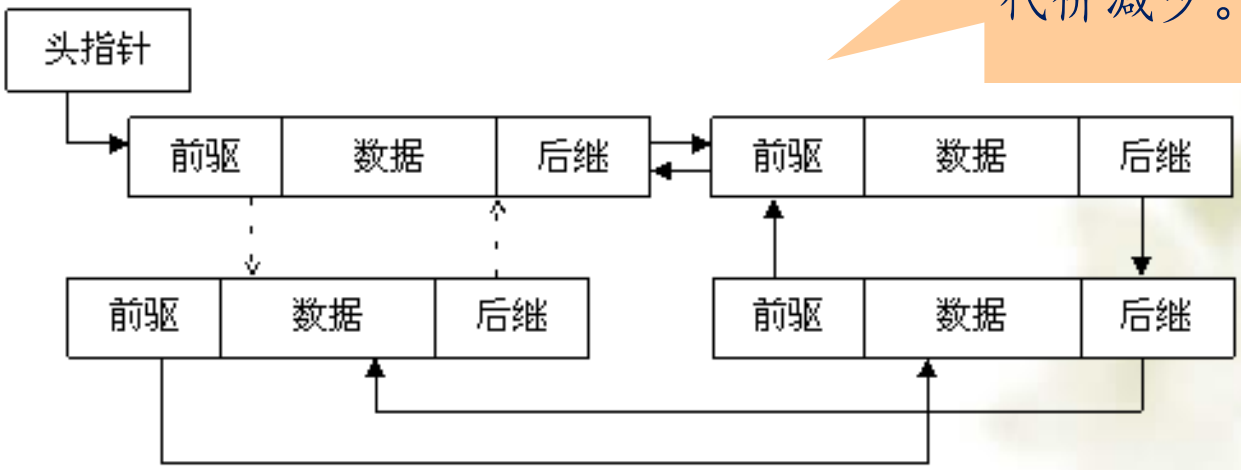
西安邮电大学

链表的演化

在C语言中，一个基本的双向链表定义如下：

```
struct my_list {
   void *mydata;
   struct my_list *next;
  struct my_list *prev;
};
```

通过前趋（prev）和后继（next）两个指针域，就可以从两个方向遍历双链表，这使得遍历链表的代价减少。



链表的演化

★如果减少一个指针域，就退化成单链表
★如果只能对链表的首尾进行插入或删除操作，就演变为队结构
★如果只能对链表的头进行插入或删除操作，就退化为栈结构
★如果前驱和后继表示左右孩子，则演变为一颗二叉树

这就说明Linux内核为什么把循环双链表作为一个基本类型。

Linux内核中链表的定义和使用

Linux内核对链表的实现方式与众不同，在链表中并不包含数据，其具体的定义如下：

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

这个链表结构常常被嵌入到其他结构中，比如：

```
struct my_list{
    void *mydata;
    struct list_head list;
    };
```

说明：list域隐藏了链表的指针特性

以struct list_head为基本对象，可以对链表进行插入、删除、合并以及遍历等各种操作，这些操作位于内核的头文件list.h中。

链表的初始化

◆**链表的声明和初始化**

内核核代码list.h中定义了两个宏：

```
#define LIST_HEAD_INIT(name) { &(name), &(name) }
 /*仅初始化*/
#define LIST_HEAD(name) struct list_head name =
                        LIST_HEAD_INIT(name) /*声明并初始化*/
```

```
static inline int list_empty(const struct list_head *head)
{
        return head->next == head;
}
```

链表的插入

◆在链表中增加一个节点

在include/linux/list.h中增加结点的函数为：

```
static inline void list_add();
static inline void list_add_tail();
```

在内核代码中，list_add()和list_add_tail()均调用__list_add()真正实现头插和尾插，函数名前加两个下划线表示内部函数。

```
static inline void list_add(struct list_head *new, struct
list_head *head)
  {
        __list_add(new, head, head->next);
  }
```

该函数向指定链表的head结点后插入new结点。因为是循环链表，而且通常没有首尾结点的概念，所以可以将任何结点传给head。若传最后一个元素给head，该函数就可以实现一个栈。

链表的插入

◆在链表中增加一个节点

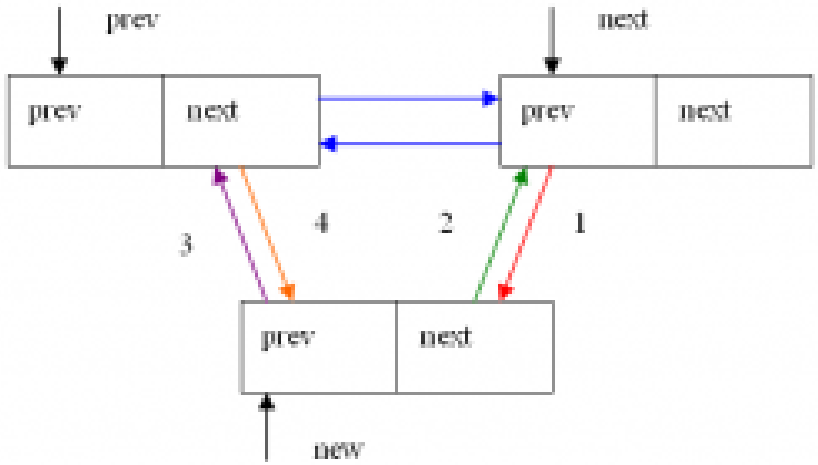
在include/linux/list.h中增加结点的函数为：

static inline void list_add();

static inline void list_add_tail();

在内核代码中，list_add()和list_add_tail()均调用__list_add()真正实现头插和尾插，函数名前加两个下划线表示内部函数。

```
static inline void __list_add(struct list_head*new,
                              struct list_head *prev,
                              struct list_head *next)
 {
      next->prev = new;
      new->next = next;
      new->prev = prev;
      prev->next = new;
 }
```



链表的插入

- list_add_tail()的内核实现：

```
static inline void list_add_tail(struct list_head *new,
struct list_head *head)
 {
        __list_add(new, head->prev, head);
 }
```

list_add_tail()函数向指定链表的head结点前插入new结点。

说明：关于static inline关键字。

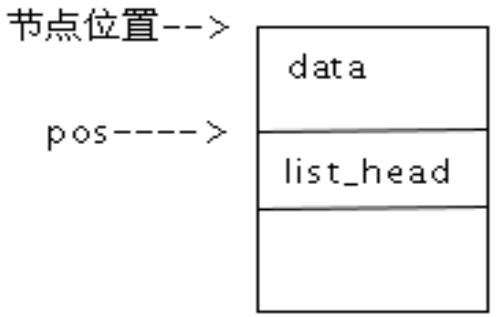
“static”加在函数前，表示这个函数是静态函数，所谓静态函数，实际上是对函数作用域的限制，指该函数的作用域仅局限于本文件。所以说，static具有信息隐藏的作用。而关键字"inline“加在函数前，说明这个函数对编译程序是可见的，也就是说编译程序在调用这个函数时就立即展开该函数。

链表的遍历

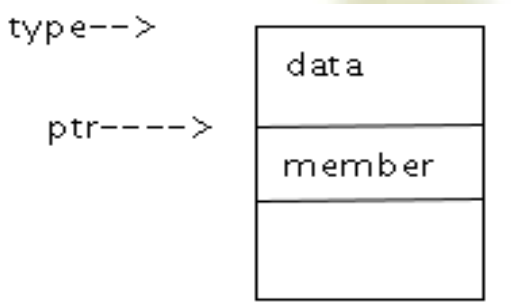
list.h中定义了如下遍历链表的宏：

```
#define list_for_each(pos, head) \
        for (pos = (head)->next; pos != (head); \
        pos = pos->next)
```

这种链表只是找到了一个个结点在链表中的偏移位置pos，如下图(a)。那么如何通过pos获得结点的起始地址，从而可以引用结点中的域呢？



(a)



(b)

链表的遍历

list.h中定义了晦涩难懂的list_entry（）宏：

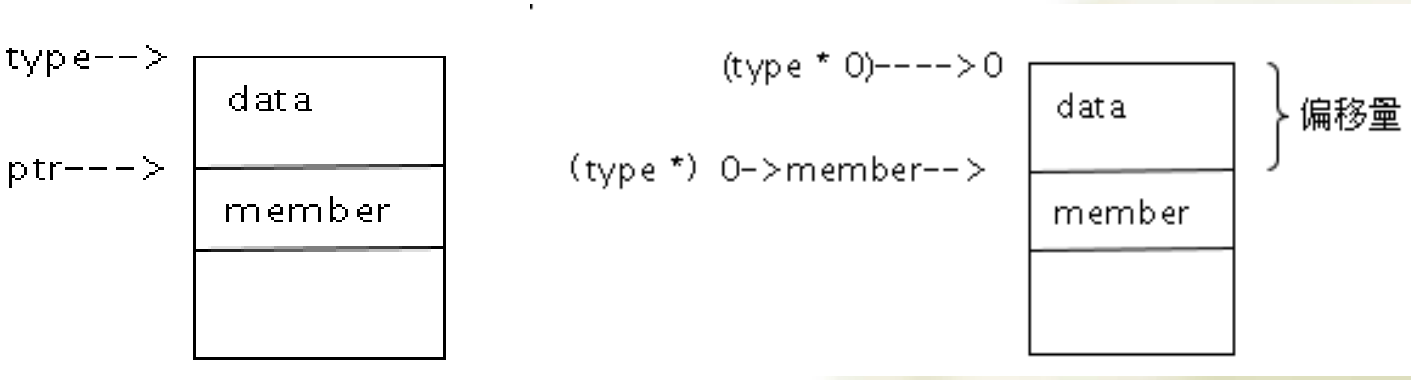
```
#define list_entry(ptr, type, member) \ ((type *)((char *)(ptr)-(unsigned long)(&((type *)0)->member)))
```

指针ptr指向结构体type中的成员member；通过指针ptr，返回结构体type的起始地址，也就是list_entry返回指向type类型的指针，如上图(b)。

Linux内核中链表的实现

为了方便大家阅读，把上面的结构体写成这样

```
((type*)(\(char*)(ptr) - (unsigned long)
(&((type*)0)->member) ))
```



（c）list_entry宏解析

Linux内核中链表的实现

((type *)0)->member把0地址强制转化为type结构的指针，再访问type结构体中的member成员，(&((type *)0)->member)获得了member在type结构中的偏移量。其中(char *)(ptr)求出的是ptr的绝对地址，二者相减，于是得到type类型结构体的起始地址。

Linux内核代码移植到用户空间

Linux 内核中不仅提供了双链表的各种接口函数，还提供了哈希表以及RCU锁的接口函数， 大约70个左右，这些接口进行适当改造后就可以移植到用户空间来使用，从而使内核经典的设计理念和代码具有可移植性。
更多的函数和宏的实现请查看include/linux/list.h中的代码。

更多函数的分析请参看Linux内核之旅：
http://www.kerneltravel.net/?page_id=568

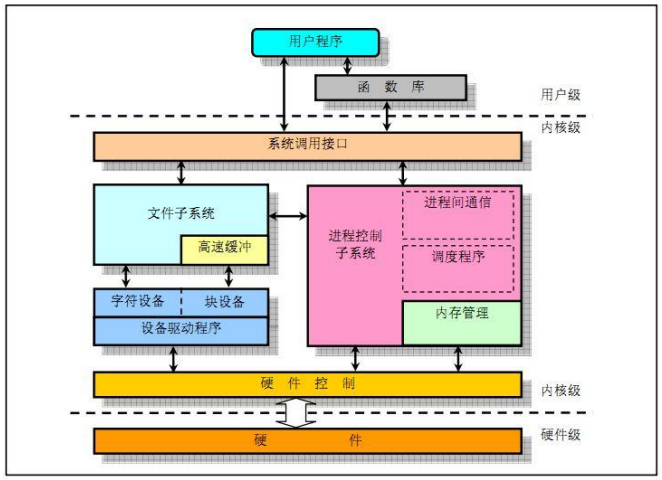
动手实践

- Linux内核之旅网站：http://www.kerneltravel.net/
- “新手上路”栏目有一系列的入门文章
- “电子杂志”栏目是关于内核研究和学习的资料，其中 第一期“走入Linux世界”涉猎了操作系统的来龙去脉后与大家携手步入Linux世界。下载代码，亲手搭建实验系统。
- 如果你希望从0开始写一个自己的操作系统，请查看毕设项目“hurlex —— x86架构的内核Demo实现”，有完整的文档和代码：http://hurlex.0xffffff.org/

Linux内核学习指南

朋友

疑问



先架构后细节

低版本理解原理，高版本理解实现

Linux内核学习指南

Linux 内核之旅
微信平台

蓝墨云班课
开放分享班391045

动手实践并总结
形成自己的智慧

谢谢大家！

THANK YOU