此文章已于 9:39:00 2016/6/21 重新发布到 MADHEX

Linux内核中链表的实现与应用

类别 [选择一个类别或键入一个新类别] ; [选择一个类别或键入一个新类别]

       链表（循环双向链表）是Linux内核中最简单、最常用的一种数据结构。

       1、链表的定义

            struct list\_head {

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

            }

           这个不含数据域的链表,可以嵌入到任何数据结构中,例如可按如下方式定义含有数据域的链表：

            struct my\_list {

                void  \* mydata;

                struct list\_head  list;

            } ;

       2、链表的声明和初始化宏

            struct list\_head 只定义了链表结点,并没有专门定义链表头.那么一个链表结点是如何建立起来的？

内核代码 list.h 中定义了两个宏：

            #defind  LIST\_HEAD\_INIT(name)    { &(name), &(name) }      //仅初始化

            #defind  LIST\_HEAD(name)     struct list\_head  name = LIST\_HEAD\_INIT(name)  //声明并初始化

            如果要声明并初始化链表头mylist\_head，则直接调用：LIST\_HEAD(mylist\_head)，之后，

mylist\_head的next、prev指针都初始化为指向自己。这样，就有了一个带头结点的空链表。

             判断链表是否为空的函数：

             static inline int list\_empty(const struct list\_head  \* head) {

                  return head->next  ==  head;

              }    //返回1表示链表为空，0表示不空

      3、在链表中增加一个结点

          （内核代码中，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 ;

            }

            list.h 中增加结点的两个函数为：

           （链表是循环的，可以将任何结点传递给head，调用这个内部函数以分别在链表头和尾增加结点）

            static inline void list\_add(struct list\_head \*new, struct llist\_head \*head)

            {

                    \_\_list\_add(new, head, head -> next) ;

            }

            static inline void list\_add\_tail(struct list\_head 8new, struct list\_head \*head)

            {

                     \_\_list\_add(new, head -> prev, head) ;

            }

            附：给head传递第一个结点，可以用来实现一个队列，传递最后一个结点，可以实现一个栈。

            static 加在函数前，表示这个函数是静态函数，其实际上是对作用域的限制，指该函数作用域仅局限

           于本文件。所以说，static 具有信息隐蔽的作用。而函数前加 inline 关键字的函数，叫内联函数，表

           示编译程序在调用这个函数时，立即将该函数展开。

    4、 遍历链表

           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)

           这种遍历仅仅是找到一个个结点的当前位置，那如何通过pos获得起始结点的地址，从而可以引用结

点的域？list.h 中定义了 list\_entry 宏：

           #define   list\_entry( ptr, type, member )  \

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

          分析：(unsigned long) ( &( (type \*)0 )  ->  member ) 把 0 地址转化为 type 结构的指针，然后获取该

          结构中 member 域的指针，也就是获得了 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，即起始结点的地址。

   5、链表的应用

         一个用以创建、增加、删除和遍历一个双向链表的Linux内核模块

1 #include <linux/kernel.h>

2 #include <linux/module.h>

3 #include <linux/slab.h>

4 #include <linux/list.h>

5

6 MODULE\_LICENCE("GPL");

7 MODULE\_AUTHOR("LUOTAIJIA");

8

9 #define N 10

10 struct numlist {

11 int num;

12 struct list\_head list;

13 };

14

15 struct numlist numhead;

16

17 static int \_\_init doublelist\_init(void)

18 {

19 *//初始化头结点*

20 struct numlist \* listnode; *//每次申请链表结点时所用的指针*

21 struct list\_head \* pos;

22 struct numlist \* p;

23 int i;

24

25 printk("doublelist is starting...\n");

26 INIT\_LIST\_HEAD(&numhead.list);

27 */\**

28  *\* static inline void INIT\_LIST\_HEAD(struct list\_head \*list)*

29  *\* {*

30  *\* list->next = list;*

31  *\* list->prev = list;*

32  *\* }*

33  *\*/*

34

35 *//建立N个结点，依次加入到链表当中*

36 for (i=0; i<N; i++) {

37 listnode = (struct numlist \*)kmalloc(sizeof(struct numlist), GFP\_KERNEL);

38 *//void \*kmalloc(size\_t size, int flages)*

39 *//分配内存，size 要分配内存大小，flags 内存类型*

40 listnode->num = i+1;

41 list\_add\_tail(&listnode->list, &numhead.list);

42 printk("Node %d has added to the doublelist...\n", i+1);

43 }

44 *//遍历链表*

45 i = 1;

46 list\_for\_each(pos, &numhead.list) {

47 p = list\_entry(pos, struct numlist, list);

48 printk("Node %d's data: %d\n", i, p->num);

49 i++;

50 }

51 return 0;

52 }

53

54 static void \_\_exit doublelist\_exit(void)

55 {

56 struct list\_head \*pos, \*n;

57 struct numlist \*p;

58 int i;

59

60 *//依次删除N个结点*

61 i = 1;

62 list\_for\_each\_safe(pos, n, &numhead.list) {

63 *//为了安全删除结点而进行的遍历*

64 list\_del(pos); *//从链表中删除当前结点*

65 p = list\_entry(pos, struct numlist, llist);

66 *//得到当前数据结点的首地址，即指针*

67 kfree(p); *//释放该数据结点所占空间*

68 printk("Node %d has removed from the doublelist...\n", i++);

69 }

70 printk("doublelist is exiting...\n");

71 }

72

73 module\_init(doublelist\_init);

74 module\_exit(doublelist\_exit);

75

参考资料：Linux操作系统原理与应用（第2版） 陈莉君、康华 编著