- 1 \$find -name "list.h"
- 2 ./btrfs/list.h

内核链表与普通链表的区别:

- 1. 普通链表当中数据域和指针域,没有做到区分。数据与指针形成了一个整体,而内核链表数据与指针是完全剥离的没有直接的关系。
 - 2. 在普通链表当中所有节点的数据都是一样的类型,而内核链表中每一个节点的类型都可以是不同的。
- 3. 普通的链表每一个节点都只能出现在一个链表当中,因为它只有一对前驱/后继指针,而内核链表可以有多对前驱/后继指针。

纯粹的内核链表(没有数据):

只是一个链表的逻辑,并没有任何的意义

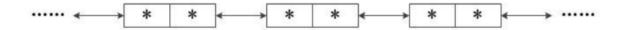


图 3-29 纯粹的链表逻辑

在数据的结构体中加入内核链表,就可以使得内核链表携带数据出现在链表当中去。

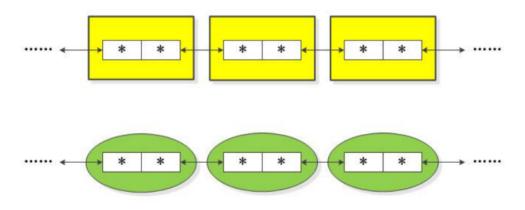


图 3-30 镶嵌了纯粹链表的节点

由于内核链表它的特性:数据与指针的完全剥离,可以使得一个节点出现在多个链表当中

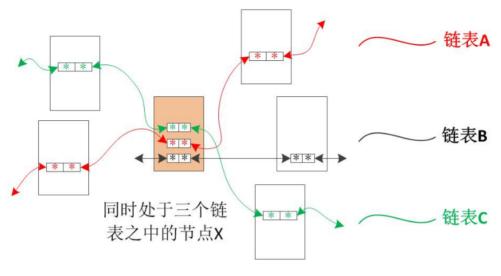


图 3-33 内核链表的优势

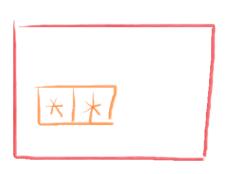
内核链表头文件:

```
1 $find -name "list.h"
2 ./btrfs/list.h
```

大结构体与小结构体:

大结构体: 包含了数据以及指针(内核链表中的小的结构体)

小结构体: 内核链表中用来指向前一个节点和后一个节点的指针



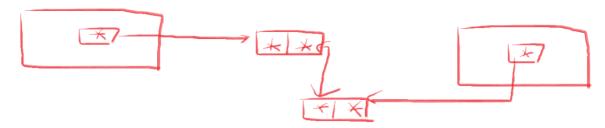
```
struct list_head {
    struct list_head *next, *prev;
};

// 节点设计
struct node
{
    Data_Type data;
    struct list_head ptr;
};
```

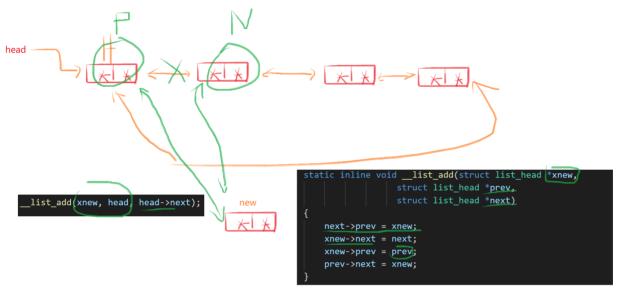
注意:

在大结构体当中小结构体必须是一个普通变量不可以使用指针。

错误示范



节点初始化:



内核链表中插入数据的思路:

把用户传进来的两个参数: 新节点 + 头节点

演变成很三个参数: 新节点 + 新节点的前驱节点 + 新节点的后继节点

```
1 /**
2 * 节点头插
3 *
4 * 使用内联函数list_add(struct list_head *xnew, struct list_head *head),
5 * 将会自动调用__list_add, 实现功能: 将节点 xnew 插入到 head 后面; 实现头插功能
6 */
7 static inline void list_add(struct list_head *xnew, struct list_head *head)
8 {
9    __list_add(xnew, head, head->next);
10 }
```

计算获得大结构体的入口地址:

```
小结构体的实际地址是已知的
                            过右边的公式计算出来差值(距离0地址)
使用已知的小结构体地址
减去差值 得到大结构体
的实际入口地址
                    小结构体的地址 大结构体的类型 小结构体的名字
                                                  假装通过0来访问大结构体中的某个(小结构体)
               #define list_entry(ptr, type, member) \
                  ((type*)((char *)(ptr) - (unsigned long)(&((type *)0)->member)))
     把最终结算的结果
     强转为大结构体指针
                         小结构体的实际地址 因为刚才获得的是
                                     一个内存地址,
                 把小结构体的类型强转为
                                     为了地址值不丢失
                 char * 使得指针加减时
                                     前转成8字节的类型
                 单位为1字节
```

```
1 P_Node kernelListInit(Data_Type data )
2
      // 申请大结构体的内存空间
3
      P Node new = calloc(1, sizeof(Node));
4
5
      //初始化数据
6
7
      new->data = data ;
8
      // 初始化小结构体(指针)
9
10
      INIT_LIST_HEAD( &new->ptr );
11
12
      return new;
13 }
```

遍历显示:

```
1
2 int display_list(P_Node head)
3 {
4     if( head->ptr.next == &head->ptr )
5     {
6         printf("该表为空!!\n");
7         return -1;
8     }
9
10     struct list_head * tmp = head->ptr.next;
```

```
11
      P_Node node = NULL ;
      for ( ; tmp != &head->ptr ; tmp = tmp->next)
12
13
          // 计算获得大结构体的入口地址
14
          node = list_entry(tmp , Node , ptr );
15
          printf("data:%d\n" , node->data);
16
       }
17
18
19
       return 0;
20 }
```

